HERIOT-WATT UNIVERSITY
EDINBURGH

DEPARTMENT OF
COMPUTER SCIENCE

ICL 4130 COBOL

B. C AIRNDUFF

September 1973

(1978)

HERIOT-WATT UNIVERSITY, EDINBURGH

DEPARTMENT OF COMPUTER SCIENCE

I C L

4 1 3 0

C O B O L

BRIAN CAIRNDUFF.

SEPTEMBER, 1973.

C O N T E N T S

		PAGE
1.0	INTRODUCTION	1
	1.1 General Description	2
2.0	BASIC FACTS	3
	2.1 Character Set	3
	2.2 Punctuation	5
	2.3 Word Formation	6
	2.4 Qualification of Names	7
	2.5 Coding Sheet	8
3.0	IDENTIFICATION DIVISION	10
4.0	ENVIRONMENT DIVISION	11
	4.1 Configuration Section	11
	4.2 Input-Output Section	11
5.0	DATA DIVISION	13
	5.1 File Description	14
	5.2 Concepts of Data Description	16
	5.3 Record Description	26
	5.4 Examples of Record Description	43
6.0	PROCEDURE DIVISION	45
	6.1 Syntax	45
	6.2 Arithmetic Expressions	48
	6.3 Input/Output Statements	49
	6.4 Data Manipulation Statements	55
	6.5 Ending Statement	63
	6.6 Arithmetic Statements	64
	6.7 Conditional Statements	72
	6.8 Branching Statements	79

PAGE

6.9 Compiler Directing Statements 86

7.0 FAULT FINDING FACILITIES 89

7.1 Trace Statement 89

7.2 Exhibit Statement 89

7.3 ON Statement 91

APPENDIX I RESERVED WORD LIST 92

APPENDIX II CONTROL LINE 94

APPENDIX III INDEX 97

1.0 INTRODUCTION

The purpose of this document is to provide the reader with a working knowledge of the version of COBOL-61 available on the University ICL 4130 computer configuration.

The COBOL compiler operates under the 4130 Commercial System which is a slave of the DES2 operating system.

The following peripherals are available for use:-

Card Reader

Paper Tape Reader

Paper Tape Punch

Line Printer

4 Magnetic Tape Units

Control Teletype

32K Word Core Storage

1.1 GENERAL DESCRIPTION

A COBOL program consists of four divisions which appear in the following order:

IDENTIFICATION DIVISION

ENVIRONMENT DIVISION

DATA DIVISION

PROCEDURE DIVISION

IDENTIFICATION DIVISION

Contains information that identifies the program being compiled and will provide useful information for documentary purposes.

ENVIRONMENT DIVISION

Contains information relating to the equipment used, as well as information concerning re-run procedures and file selection.

DATA DIVISION

Provides the compiler with a description of the file structure, working storage and constants used.

PROCEDURE DIVISION

Consists of the steps that the computer is to follow during the data processing operation. These steps are written as statements making up sentences, paragraphs and sections.

2.0 BASIC FACTS

The information contained in this section will provide a useful reference to the dynamics of COBOL program writing.

2.1 CHARACTER SET

The complete character set which is permitted in COBOL consists of the following 52 characters:

Digits 0 through 9

Letters A through Z

SPECIAL CHARACTERS

+	Plus sign
-	Minus sign or hyphen
	Blank or space
*	Cheque protection symbol, asterisk
/	Slash
=	Equal sign
>	Inequality sign (greater than)
<	Inequality sign (less than)
£	Pound sign
,	Comma
.	Fullstop or decimal point
'	Quotation mark
(	Left parenthesis
)	Right parenthesis
;	Semicolon
&	Ampersand

Of the previous set, the following characters are used for words:

'	Quotation mark
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Fullstop
;	Semicolon

The following characters are used in arithmetic expressions:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The following characters are used in relation tests:

<	Inequality sign (less than)
>	Inequality sign (greater than)
=	Equality sign

All of the preceding characters are contained in the COBOL character set. In addition, the programmer may use, as characters in non-numeric literals, any characters (except newline, tabulate and quotation mark) included in in-shift 4100 internal code; however, such characters may be unacceptable to COBOL for computers other than those of the 4100 range.

2.2 PUNCTUATION

The following rules of punctuation apply in writing COBOL source programs:

1) When any punctuation mark is indicated in a format in this publication, it is mandatory.

2) A fullstop, semicolon, or comma, when used, must not be preceded by a space, but must be followed by a space.

3) A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded immediately by a space.

4) At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space except in non-numeric literals.

5) Except in the case of a unary operator, an arithmetic operator or an equal sign must always be preceded by a space and followed by another space. A unary operator may be preceded by a left parenthesis.

6) When the fullstop, or comma, or arithmetic operator characters are used in the PICTURE clause as editing characters, they are governed by the rules for report items only.

7) A comma may be used as a separator between successive operands of a statement.

8) A comma or a semicolon may be used to separate a series of clauses.

9) A semicolon or the word THEN may be used to separate a series of statements.

2.3 WORD FORMATION

A word is composed of a combination of not more than 30 characters, chosen from the following set of 37 characters:

0 through 9 (digits)
A through Z (letters)
- (hyphen)

A word must not begin or end with a hyphen. A word is ended by a space, or by proper punctuation. A word may contain one or more embedded hyphens; consecutive embedded hyphens are also permitted. All words in COBOL are either reserved words, which have pre-assigned meanings in COBOL, or programmer-supplied names. Each type of word is discussed in the section of this publication in which it is first required.

2.4 QUALIFICATION OF NAMES

Every names used in a COBOL source program must be unique within the source program, in either of two ways:

(a) because no other name has the identical spelling

(b) because the name exists within one of a number of hierarchies of names, so that the name can be made unique by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers when this method of ensuring uniqueness is used. The process is called qualification.

The following rules apply to the qualification of names:

1) The word OF or IN must precede each qualifier and the names must appear in ascending order of hierarchy.

2) A qualifier must be a higher level and within the same hierarchy as the name it is qualifying.

3) The same name must not appear at two levels in a hierarchy, as it would appear to qualify itself.

4) The highest level qualifier must be unique. Each qualifying name must be unique at its own level within the hierarchy of the immediately higher qualifier.

5) Qualification is permitted, even when not required.

6) Qualifiers must not be subscripted, although the entire qualified name may be subscripted.

7) The total number of characters in a series of qualifiers, together with the qualified name, may not exceed 200.

8) No matter what qualification is available, a procedure-name must not be the same as a data-name.

2.5 COBOL Coding Sheet

The COBOL coding sheet provides a standard format for writing COBOL source programs.

Conceptually, one blank is assumed to be appended after column 72 on every line of a program sheet in the Cobol Text, except where a non-numeric literal spans more than one line.

SEQUENCE NUMBER: (Columns 1-6)

The sequence number must consist only of digits. The sequence number has no effect on the source program and need not be written, but is provided to facilitate source code editing prior to compilation. If the programmer supplies sequence numbers, the compiler will indicate any errors in sequences. If these columns are blank, no sequence check will be made.

CONTINUATION INDICATOR: (Column 7)

A hyphen may be placed in this column for the continuation of non-numeric literals (see the subsection, "Continuation of Non-numeric Literals" below).

SOURCE PROGRAM TEXT: (Columns 8-72)

The source program text is written in these columns.

IDENTIFICATION CODE: (Columns 73-80)

These columns may be used to refer to a group of source lines. Any character from the COBOL character set may be used, including the blank. The identification code has no effect on the object program produced by the compiler.

Margin Restrictions

There are two margins on the COBOL program sheet: Margin A (Column 8), and Margin B (Column 12).

The name of a division must begin in Margin A. The division-name must be followed by a space, the word DIVISION, and a fullstop. This entry must appear on a line by itself.

A section-name must begin in Margin A, be followed by a space, the word SECTION, optionally a priority number and then a fullstop. This entry must appear on a line by itself, except when an INCLUDE statement follows it. A paragraph-name must also begin in Margin A and must be followed immediately by a fullstop and a space. Statements may start on the same line in or after Margin B. Succeeding lines of the paragraph must start in or after Margin B.

The level indicator (FD) of the File Description entry in the Data Division, must begin in Margin A. Names and clauses within these entries must begin in or after Margin B. The level numbers (01-49, 77, 88) of data description entries may begin before Margin B, however, the rest of the entry (data-names and/or clauses) must not begin before Margin B.

CONTINUATION OF NON-NUMERIC LITERALS

When a non-numeric literal is too long to fit on one line of a coding sheet, the following rules apply.

(1) A hyphen is placed in column 7 of the continuation line.

(2) A quotation mark is placed in or after Margin B preceding the continuation of the literal.

(3) All spaces at the end of the continued line, and any spaces following the quotation mark in the continuation line and preceding the final quotation mark of the literal, are taken as part of the literal.

3.0 IDENTIFICATION DIVISION

The format of the Identification Division is:-

```
PROGRAM-ID.      Program-name.

AUTHOR.          sentence.

INSTALLATION.    sentence.

DATE-WRITTEN.    sentence.

SECURITY.        sentence.

REMARKS.         sentence.
```

Program-name identifies the program, and must consist of 8 characters. The first must be a letter, and others may be taken from the set 0-9, A-Z. All the COBOL words must begin in Margin A. Sentences must begin in or after Margin B, and be terminated by a full stop.

N.B. All entries with the exception of PROGRAM-ID are optional.

A typical Identification Division could look like this.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    TESTPROG.
AUTHOR.        A N OTHER.
INSTALLATION.  H-W.
DATE-WRITTEN.  18TH MAY 1972.
SECURITY.      NONE.
REMARKS.       THIS PROGRAM READS CARDS, CREATING AN OUTPUT FILE
               ON TAPE OF VALID RECORDS AND OUTPUTTING INVALID
               CARDS TO THE PRINTER.
```

4.0 ENVIRONMENT DIVISION

The Environment Division is divided into two sections:

the CONFIGURATION SECTION

and the INPUT-OUTPUT SECTION.

4.1 The format of the CONFIGURATION Section is:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  ICL-4130.
OBJECT-COMPUTER.  ICL-4130 SIZE 30000 WORDS.
```

All the COBOL words must begin in Margin A. Other entries must begin in or after Margin B, and be terminated by a full stop.

N.B. All entries with the exception of CONFIGURATION SECTION are optional.

4.2 The format of the INPUT-OUTPUT Section is:

```
                INPUT-OUTPUT SECTION.
                FILE-CONTROL.
SELECT filename ASSIGN TO CREADER      Integer.
                          PREADER      Integer.
                          PPUNCH       Integer.
                          MTAPE        Integer.
                          PRINTER120   Integer.
```

The option CREADER assigns a file to the CARD READER.

The Integer value is 0.

The option PREADER assigns a file to the PAPER TAPE READER.

The Integer value is 0.

The option PPUNCH assigns a file to the PAPER TAPE PUNCH.

The Integer value is 0.

The option PRINTER120 assigns a file to the LINEPRINTER.

The Integer value is 0.

The option MTAPE assigns a file to the MAGNETIC TAPE UNIT.

The integer value may range from 0-7, the usual number being 3 or 4.

If no files are required, all the entries of this section may be omitted.

The SELECT sentence must begin with the word SELECT, and must start at Margin B.

A sentence must be given for each file required, and all sentences terminated by a full stop.

Each filename must be unique within a program, and must have a File Description (FD) entry in the Data Division.

An example of an INPUT-OUTPUT SECTION:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT CARDFILE       ASSIGN TO CREADER 0.
    SELECT MASTER-FILE    ASSIGN TO MTAPE 3.
    SELECT PRINTOUT       ASSIGN TO PRINTER120 0.
```

5.0 DATA DIVISION

The DATA DIVISION comprises of two sections:

FILE SECTION

and WORKING-STORAGE SECTION.

Storage space is allocated to Files in the FILE SECTION, all other storage requirements are allocated to the WORKING-STORAGE SECTION.

The format of the DATA DIVISION is:-

```
DATA DIVISION.

FILE SECTION.

File Description entries

Record Description entries

WORKING-STORAGE SECTION.

Record Description entries.
```

5.1 FILE DESCRIPTION

File Description entries supply information on the physical structure of a file.

The possible entries are:

A B

FD File-name

RECORDING MODE IS mode

BLOCK CONTAINS Integer RECORDS

VALUE OF ID IS Literal

RECORD CONTAINS Integer CHARACTERS

LABEL RECORDS ARE STANDARD or OMITTED

DATA RECORD IS record-name-1, record-name-2, ---------.

N.B. A fullstop occurs at the end of the DATA RECORD entry ONLY.

FD is written in Margin A. All other entries are written within Margin B.

5.1.1 The RECORDING MODE IS clause applies to the type of record in the file. Mode must be either the character F, or the character V. (Fixed or Variable.) If this clause is omitted, V is assumed.

5.1.2 The BLOCK CONTAINS Integer RECORDS clause applies to Magnetic Tape files only. If the clause is omitted, the block will consist of only one record.

N.B. Block length = 511 Words. Allow 1 word per block for a Word-count.

5.1.3 The VALUE of ID clause applies to Magnetic Tape files only. The length of literal must not exceed 16 characters. When literal is used for output files, the name in the header takes its value from literal. For input files, the file opened has a name of value literal. If the clause is omitted, the file header contains File-name as the name of the file.

5.1.4 The RECORD CONTAINS clause specifies the size of the logical records contained in a file.

5.1.5 The LABEL RECORDS clause must be included for all file descriptions. It specifies the presence or absence of standard labels on a File. For files assigned to Magnetic Tape, the STANDARD option is specified, and the OMITTED option specified for all other files.

5.1.6 The DATA RECORD IS clause specifies the names of all the logical records in a File, i.e. all the data-names with an 01 level-number.

Example of File Description:-

```
FD      MASTER-FILE
        RECORDING MODE IS F
        BLOCK CONTAINS 20 RECORDS
        VALUE OF ID IS 'MASTER-FILE'
        RECORD CONTAINS 100 CHARACTERS
        LABEL RECORDS ARE STANDARD
        DATA RECORD IS DETAIL-REC, TOTALS-REC.
```

5.2 CONCEPTS OF DATA DESCRIPTION

5.2.1 CONCEPT OF LEVELS

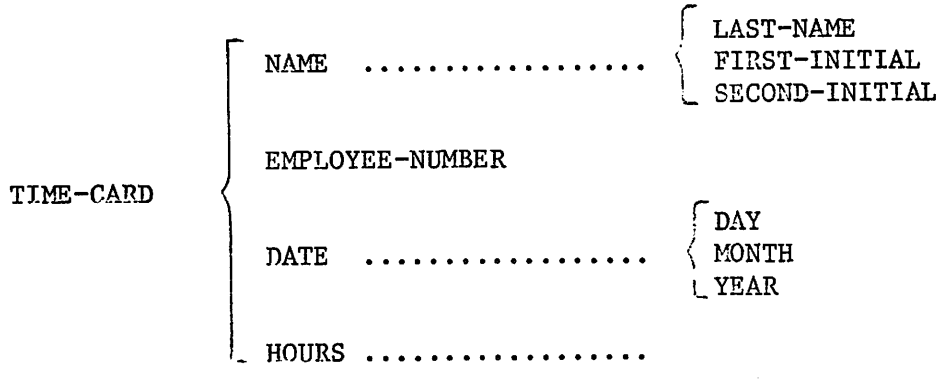
Level indicators are used to show how data items are related to each other. The most inclusive grouping of data is the file, for which the level indicator is FD.

The contents of a file are divided into logical records, with a level number of 01 specifying a record. Subordinate data items that constitute a logical record are grouped in a hierarchy and identified with level numbers 02 to 49.

The level number 77 identifies a special type of entry in the Working-Storage Section.

The level number 88 is used to define a condition-name associated with a value.

Levels allow reference to named parts or sub-divisions of a record. Once specified, a sub-division may be further sub-divided to permit even more detailed data referencing. This is illustrated by the following weekly time-card record, which is divided into four major items: name, employee-number, date and hours, with more specific information appearing for name, and date.



Items that contain subdivisions are known as group items. Sub-divisions of a record, that are not further subdivided, are called elementary

items. When a PROCEDURE DIVISION Statement refers to a group item, the reference applies to the area reserved for the entire group.

(1) Less inclusive groups are assigned higher-level numbers.

(2) Level numbers of items within groups need not be consecutive.

(3) A group includes all groups and elementary items described under it until a level number less than or equal to the level number of the group is encountered.

(4) Separate entries are written in the source program for each level.

To illustrate level numbers and group items, the weekly timecard record in the previous example may be described by Data Division entries having the level numbers and date-names illustrated in Figure 1. The spacing is optional, but desirable for legibility.

The level number 77 identifies a Record Description entry for independent WORKING-STORAGE items. The level number 77 cannot appear in the FILE SECTION.

The level number 88 assigns a condition-name.

FIGURE 1 - EXAMPLE OF DATA LEVELS:

```
01  TIME CARD.
    02  NAME.                                          -  GROUP
        03  LAST-NAME          PICTURE X(20).         -  ELEMENTARY
        03  FIRST-INITIAL      PICTURE X.             -  ELEMENTARY
        03  SECOND-INITIAL     PICTURE X.             -  ELEMENTARY
    02  EMPLOYEE-NUMBER        PICTURE 9(5).          -  ELEMENTARY
    02  DATE.                                         -  GROUP
        03  DAY                PICTURE 99.            -  ELEMENTARY
        03  MONTH              PICTURE 99.            -  ELEMENTARY
        03  YEAR               PICTURE 99.            -  ELEMENTARY
    02  HOURS                  PICTURE 99V99.         -  ELEMENTARY
```

01 levels are written in Margin A - all other levels are written within

Margin B. For the sake of clarity, each level is indented according to its number, i.e. level 02 would be written from column 12, 03 from column 16, 04 from column 20 etc..

5.2.2 DATA-NAMES

A DATA-NAME is a word assigned by the user to identify a data item used in a program.

(1) A data-name always refers to a kind of data, not to a particular value, as the item referred to usually assumes a number of different values during the course of a program.

(2) A data-name must contain at least one alphabetic character.

(3) A data-name, or the key word FILLER, must be the first word after the level number in each Record Description entry, as shown in the following general format.

Level-number $\{^{\text{data-name}}_{\text{FILLER}}\}$ (description for elementary item).

Data-name is the defining name of the entry and is the means by which references to the associated data area (containing the value of a data item) are made.

(4) If some of the characters in a record are not referred to, the data description of these characters need not include a data-name. In this case, the reserved word FILLER is written in lieu of data-name. FILLER need not be unique and may be assigned a pre-set value. It may not occur at level 01.

(5) If the same data-name is assigned to more than one item in a program, the data-name must be qualified in all references to it in the Procedure Division.

(6) A data-name is qualified by writing either IN or OF after it, followed

by the name of one or more groups in which it occurs, or the name of the file in which it is contained. A highest level qualifier must, however, be unique.

(7) Any combination of qualifiers that will ensure uniqueness may be used. More qualifiers may be used than are absolutely necessary, for example, in Figure 1, if YEAR OF DATE is needed to make YEAR unique, YEAR OF DATE IN TIME-CARD is also permitted, as is YEAR IN TIME-CARD.

(8) A data-name may not be subscripted when it is used as a qualifier. However the entire qualified data-name may be subscripted.

5.2.3 LITERALS

A literal is a constant that is not identified by a data-name in a program, but is completely defined by its own identity. A literal is either non-numeric (alphabetic or alphanumeric) or numeric.

5.2.3.1

Non-Numeric Literals

A non-numeric literal must be bound by quotation marks and may consist of any combination of characters in the 4100 code, except, quotation mark. All spaces enclosed by the quotation marks are included as part of the literal. A non-numeric literal must not exceed 120 characters in length, excluding the quotation marks.

The following are examples of non-numeric literals:

```
'EXAMINE CLOCK NUMBER'
'12565'
'PAGE 144 MISSING'
```

5.2.3.2

Numeric Literals

A numeric literal must contain at least one, and not more than 12, digits. It may consist of the characters 0 to 9, and the plus sign, the minus sign and the decimal point. It may contain only one sign character and only one decimal point. The sign, if present, must appear as the leftmost character in the literal. If a literal is unsigned, it is assumed to be positive. A decimal point may appear anywhere within the literal, except as the rightmost character. If a literal does not contain a decimal point, it is considered to be a whole number.

The following are examples of numeric literals:

```
 1506798
+12572.6
-256.75
.16
```

5.2.4 FIGURATIVE CONSTANTS

A figurative constant is a special form of literal. It represents a value to which a standard data-name has been assigned. A figurative constant must NOT be bound by quotation marks.

ZERO may be used as either a numeric or a non-numeric literal. All other figurative constants are non-numeric. The singular and plural forms of figurative constants are equivalent and may be interchanged.

ZERO ZEROS ZEROES	Represent one or more zeros
SPACE SPACES	Represent one or more blanks or spaces
HIGH-VALUE HIGH-VALUES	Represent one or more occurances of the highest value in the computer's collating sequence, which for the 4100 is octal 77 or shift-in. A one-character field, with value HIGH-VALUE, may be used to operate as a delete-code or an end-of-record-code.
LOW-VALUE LOW-VALUES	Represent one or more occurences of the lowest value in the computer's collating sequence, which for the 4100 is octal 00, or space.
ALL 'character'	Represents one or more occurences of the single 4100 Code character bounded by quotation marks. Character may not be a quotation mark.
QUOTE QUOTES	Represents the character ' . Note that the use of the word QUOTE to represent the character ' at run time is not equivalent to the use of the symbol ' to bound a non-numeric literal.

When a figurative constant is used in such a way that the exact number of characters required cannot be determined by the compiler, only one character is generated. For example, the statement DISPLAY ZEROS would produce one zero character.

5.2.5 CONDITION-NAMES

The format of the condition-name entry is

88 condition-name VALUE IS literal

Condition-name is a name assigned by the user to a value that may be assumed by a data item, which is known as a conditional variable. It must be formed according to the rules for data-name formation (see 5.2.2) and be given a level number of 88. A level 88 entry must be preceded by either another level 88 entry (in the case of several consecutive condition-names pertaining to an elementary item), or by an elementary item.

Every condition-name refers to an elementary item (conditional variable). Condition-name is used in the PROCEDURE DIVISION in place of a simple relational condition.

Condition-name may refer to an elementary item requiring subscripts. In this case, condition-name, when written in the Procedure Division, must be subscripted according to the requirements of the associated elementary item.

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable.

Figure 2 shows an example of Data Division entries and a Procedure Division statement that might be written using level 88 and the condition-name test.

FIGURE 2 - CONDITION-NAME EXAMPLE

PART OF DATA DIVISION

```
01   TIME-CARD.

     02   NAME                                  PICTURE X(20).
     02   PAY CODE                              PICTURE 9.

          88   MONTHLY                          VALUE 1.
          88   HOURLY                           VALUE 2.
          88   SUBCONTRACTOR                    VALUE 3.

     02   SALARY                                PICTURE 9(4).
     02   RATE-PER-HOUR REDEFINES SALARY        PICTURE 9V999.
     02   PER-DIEM REDEFINES RATE-PER-HOUR      PICTURE 99V99.
```

PART OF PROCEDURE DIVISION

```
IF   HOURLY COMPUTE GROSS                  =   40 * RATE-PER-HOUR
     ELSE IF MONTHLY COMPUTE GROSS         =   SALARY/4.334
     ELSE IF SUBCONTRACTOR COMPUTE GROSS =   5 * PER-DIEM
     ELSE PERFORM ERROR-PROCESS.
```

5.2.6 DATA REPRESENTATION

5.2.6.1

Group Items

A group item is defined as one having further sub-divisions, so that it contains one or more elementary items.

(1) A group item may contain other groups.

(2) An item is a group item if its level number is less than the level number of the immediately following item, unless the latter level is 88.

(3) If an item is not a group item, it is an elementary item, or it is a condition-name in the case of level 88.

5.2.6.2

Elementary Items

An elementary item is a data item containing no subordinate items.

5.2.6.3

Alphabetic Items

An alphabetic item may contain any combination of the characters A to Z and space. Alphabetic characters are packed four to a 4100 word.

5.2.6.4

Alphanumeric Item

An alphanumeric item consists of any combination of characters in in-shift 4100 code. Alphanumeric characters are packed four to a 4100 word.

5.2.6.5

Numeric Display Item

A numeric display item consists of the characters 0 to 9. The last character is overpunched to show the sign of the item, such that characters P to Y represent 0 to 9 if the item is negative. Numeric fields are restricted to a maximum of 12 characters, packed four to a 4100 word.

5.2.6.6

Numeric Computational Item

A numerical computational item is held in binary form, but is represented in the source program by the characters +, - and 0 to 9. It occupies one or two 4100 words, corresponding to decimal lengths of 1 to 6 digits and 7 to 12 digits respectively.

5.3 RECORD DESCRIPTION

5.3.1 RECORD DESCRIPTION ENTRY

A record description entry specifies the characteristics of each item in a data record.

Each entry consists of:

LEVEL NUMBER

DATA NAME

A SERIES OF INDEPENDENT CLAUSES

FULL STOP

The general format of a Record Description entry is:

```
LEVEL NO.  DATA NAME OR [REDEFINES-clause] [PICTURE-clause]
           FILLER
[OCCURS-clause] [VALUE-clause] [JUSTIFIED RIGHT] [USAGE-clause].
```

Examples of the various combinations of the clauses are given in Section 5.4.

5.3.2 PICTURE CLAUSE

The PICTURE clause gives a detailed description of an ELEMENTARY level data item.

The general format of the PICTURE clause is

PICTURE IS [A | X | 9 | Report form]

If the DECIMAL-POINT IS COMMA clause is present (in the Special-Names

paragraph of the Environment Division), the rules for fullstop and comma are reversed in the PICTURE clause description below.

Wherever a string of characters is specified, the shorthand form left parenthesis, total number of occurences, right parenthesis, may be used.

5.3.2.1

Alphabetic Option - A

Option must be composed of a string of characters A. An A indicates that the character position always contains a space or one of the 26 letters of the English Alphabet.

5.3.2.2

Alphanumeric Option - X

Option must be composed of a string of characters X. An X indicates that the character position may contain any character from 4100 in-shift internal code.

5.3.2.3

Numeric Option - 9

The PICTURE of a numeric item may contain a valid combination of the following characters.

CHARACTER	MEANING
9	The character 9 indicates that the position contains a digit.
V	The character V indicates the position of an assumed decimal point. Since a numeric item cannot contain an actual decimal point, an assumed decimal point is used to provide the compiler with information concerning the scaling alignment of items involved in computations. Storage in the item is never reserved for the assumed decimal point.

CHARACTER	MEANING
S	The character S indicated the presence of a plus or minus sign. If used, it must be the left-most character of the numeric-form. The presence of S is required where USAGE IS COMPUTATIONAL has been specified. For a DISPLAY item used as input, an S must be written only if the item contains a sign. For DISPLAY items developed by the execution of COBOL statements, the compiler will generate a sign only if an S is written in the picture clause. A minus sign is indicated by an overpunch of the last character of the item.
P	The character P represents a numeric digit position for which storage is not reserved and which is always treated as if it contained a zero. P (or a group of P's) is used to scale the location of an assumed decimal point. For example, an item composed of the digits 123 would be treated by an arithmetic procedure statement as 123000 if its PICTURE were 999PPPV; or as .000123 if its PICTURE were VPPP999. The character V may be used or omitted as required. When used, V must be placed in the position of the assumed decimal point, to the left or right of the P or P's that have been specified.

The maximum length of a computational item is 12 digits. An item of 12 to 7 characters will use two words and below 7, one word.

5.3.2.4

Report-form Option

This option represents a report item. The characters that may be combined to describe a report item are as follows:

9 V P . Z * CR DB , 0 B £ + -

The characters 9, V and P have the same meaning as for a numeric item. The other characters are editing characters.

CHARACTER	MEANING
.	The decimal point character specifies that an actual decimal point is to be inserted in the indicated position and that the source item is to be aligned accordingly. Numeric character positions to the right of a decimal point in a PICTURE must consist of characters of one type (that is, * or Z or 9 or £ or + or -).
Z	This character is the zero suppression character. Each Z in a PICTURE represents a digit position. Leading zeros, to be placed in positions defined by Z, are suppressed, leaving the position

CHARACTER	MEANING
	blank. Zero suppression ends when the actual decimal point, or V, is reached. Z may appear to the right of a decimal point only if all digit positions are represented by Zs. A Z cannot appear anywhere to the right of a 9. The PICTURE ZZZ.ZZ is equivalent to a combination of the BLANK clause and the PICTURE ZZZ.99.
*	The asterisk is the "cheque protection" replacement character, which is similar to Z, except that leading zeros are replaced by asterisks instead of space. An asterisk must not appear anywhere to the right of a 9. The BLANK clause must not be applied to an item that has an asterisk in its PICTURE.
CR DB	CR and DB are credit and debit symbols and, if used, must appear at the end of the PICTURE. These symbols occupy two character positions and specify that the symbol is to appear if the value of the item is negative. If the value is positive or zero, spaces will appear instead.
, 0 B	The comma, zero and B (indicating a space) are insertion characters. Each of these characters is counted in the size of the data item. They do not represent digit positions, but the actual occurence of such a character. The presence of zero suppression (Z) or check protection (*) indicates the suppression of leading insertion characters, with replacement by spaces or asterisks. The comma, zero and B may also appear in conjunction with a floating string, as described in the following text.
£ + -	The character £ or + or - may appear in a PICTURE, either singly (fixed) or in a floating string. As a fixed sign control character, the + or - must appear as either the first or last symbol in the PICTURE, but not both. The plus indicates that a plus or minus sign is generated in the appropriate position corresponding to the algebraic sign of the item. The minus indicates that blank or minus is generated in the appropriate position. When used as fixed insertion characters these characters may appear only once in a PICTURE.

Other rules for a report item PICTURE are as shown below.

(1) The use of one type of floating string precludes the use of any other floating string.

(2) There must be at least one digit position character.

(3) If no 9 appears, BLANK WHEN ZERO is implied, unless * appears in all the numeric positions.

(4) The appearance of a floating sign string, or fixed plus or minus insertion characters, precludes the appearance of any other of the sign control insertion characters (+ - CR DB).

(5) The characters in a PICTURE to the left of a decimal point (or anywhere in the PICTURE, if no decimal point is given), excluding the characters that comprise a floating string, are subject to the following restrictions:

(a) Z must not appear to the right of * or 9 or a floating string.

(b) * must not appear to the right of 9 or Z or a floating string.

(6) The characters to the right of a decimal point, up to the end of a PICTURE, excluding the fixed insertion characters + - CR DB, are subject to the following restrictions:

(a) Only one type of digit position character may appear. That is, asterisk, Z, 9 and floating-string characters £ + - are mutually exclusive.

(b) If any of the numeric character positions, to the right of a decimal point is represented by + or - or £ or Z or *, then all the numeric character positions in the PICTURE must be represented by the same character.

(7) A floating string must begin with at least two consecutive appearances of the + or - or £ character.

(8) The PICTURE character 9 must never appear to the left of a floating or replacement character.

(9) The floating or replacement characters + - Z £ * must not be mixed in a PICTURE description. They may appear with fixed characters as described below.

(a) * or Z with fixed £.

(b) £ (fixed or floating) with fixed rightmost + or -.

(c) * or Z with fixed leftmost + or -.

(d) * or Z with fixed rightmost + or -.

(10) A PICTURE character string must not end with a decimal point, or comma, followed by a space, although it may be followed by semi-colon and space.

5.3.2.5

Floating Strings

A floating string is defined as a leading, homogenous series of £ or + or -, or a string composed of one such character interrupted by one or more insertion characters (,0B) and/or V or actual decimal point.

For example:

```
£ £, £££, £££
++++
--, ---, --
£££B£££
+(8)V++
```

A floating string containing n+1 occurences of £ or + or - defines n digit positions. When moving a numeric value into a report item, the appropriate character floats, so that the developed report item has exactly one actual £ or + or - immediately to the left of the most significant non-zero digit. Blanks are placed in all character positions to the left of the single developed £ or + or -. If the most significant digit appears to the right of positions defined by a floating string, then the developed item contains £ or + or - in the rightmost position of the floating string. The presence of an actual or implied decimal point in a floating string causes all digit positions to the right of the point to be treated as though they were indicated by the PICTURE character 9. A BLANK WHEN ZERO clause is also assumed.

A floating string need not constitute the entire PICTURE of a report item.

When B, comma, or zero appear to the right of a floating string, the string character floats through these characters to be as close to the leading digit as possible.

An embedded B, comma or zero in a floating string indicates that the character is to appear in the indicated position, unless the position immediately precedes the leading significant digit. These characters in a PICTURE need not appear singly, thus, ££BB£££ is a valid PICTURE for a report item.

5.3.2.6

Additional Notes On The Picture Clause

(1) A PICTURE clause must only be used with an elementary item.

(2) An integer enclosed in parentheses and following

A X 9 Z * 0 - B, £ +

indicates the number of consecutive occurences of the character.

(3) All characters, except P V and S are included in the total size of a data item. CR and DB occupy two character positions.

(4) The PICTURE clause must not contain more than 30 format characters. For example, PICTURE A(79) uses five format characters.

(5) A PICTURE must consist of at least one of the characters A X 9 * Z or at least two consecutive appearances of the + or - or £ character.

(6) The characters S . V CR and DB may appear only once in a PICTURE and CR and DB may not both appear in the same PICTURE.

(7) An item may possess only one plus or minus sign.

5.3.3 USAGE CLAUSE

The USAGE clause describes the form in which data is held.

The format of the USAGE clause is

```
USAGE IS [ DISPLAY       ]
         [ COMPUTATIONAL ]
```

The USAGE clause may be written at any level. At group level, it applies to every elementary item in the group. The usage of an elementary item must not conflict with the usage explicitly stated for a group to which the item belongs.

The DISPLAY option specifies that the item is stored in character form, packed four to a 4100 word.

The COMPUTATIONAL option specifies a binary data item occupying one or two words corresponding to specified decimal lengths of 1-6 digits, and 7-12 digits respectively.

If USAGE is not specified, the usage of an item is assumed to be DISPLAY.

5.3.4 VALUE CLAUSE

This clause either defines condition-name values or the initial value of working-storage items.

The format of the VALUE clause is

VALUE IS literal

e.g. VALUE IS 120.

VALUE IS 'THIS TEXT IS STANDARD'.

The size of a literal given in a VALUE clause must not be greater than the size of the item, as given in the PICTURE clause, with the provision that the literal must also include leading or trailing zeros. The positioning of the literal within a data area is the same as that which would result from specifying a MOVE of the literal to the data area. The type of literal written in a VALUE clause must correspond to the type of data item.

When an initial value is not specified, no assumption should be made regarding the initial contents of an item in working-storage.

The value clause may only be specified for elementary items, other than report items.

In the File Section, the VALUE clause may only appear with a level 88 item.

The VALUE clause must not be written in a Record Description entry that has an OCCURS or REDEFINES clause, or in an entry that is subordinate to such an entry. However, an 88 level entry containing a VALUE clause may be subordinate to an entry containing an OCCURS or REDEFINES clause.

5.3.5 JUSTIFIED RIGHT CLAUSE

This clause may only be written for an elementary alphabetic or alphanumeric item.

The format of this clause is

JUSTIFIED RIGHT

When JUSTIFIED RIGHT is specified for non-numeric data, the rightmost character of a source field is placed in the rightmost position of the receiving field. Characters are inserted in the receiving field from right to left, until it is full. If the source field is longer than the receiving field, truncation occurs when the receiving field is full. If the source field is shorter, the unfilled positions are filled with spaces.

e.g.

SOURCE FIELD		RECEIVING FIELD		
PICTURE	VALUE	PICTURE	VALUE	(AFTER MOVE)
X(5)	ABCDE	X(4)	BCDE	
X(3)	XYZ	X(5)	XYZ	

5.3.6 REDEFINES CLAUSE

This clause specifies that a single area is to contain different data items, or the clause provides an alternative description of the same data.

The format of the REDEFINES clause is:

data-name-1 REDEFINES data-name-2

In the following text, data-name-1 refers to the data name defining the entry.

The REDEFINES clause must be the first clause following data-name-1. When an area is redefined, all descriptions of the area remain in effect. Thus if B and C are two separate items that share the same storage area due to redefinition, the procedure statements MOVE X TO B or MOVE Y TO C could be executed at any point in the program.

The following rules govern the use of the REDEFINES Clause.

(1) The REDEFINES clause must not be used for logical records (i.e. level 01) associated with a single file, since redefinition already exists, by implication.

(2) The level number of data-name-2 must be identical to that of data-name-1.

(3) Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

(4) The entries giving the additional description of the area must immediately follow the original description. However, other descriptions that redefine the same area may intervene.

(5) If data-name-1 is described as COMPUTATIONAL, data-name-2 must start on a word boundary.

(6) REDEFINES clauses may be nested, i.e. they may be specified for items within the scope of an area being redefined.

(7) Entries containing, or subordinate to, a REDEFINES clause must not contain any VALUE clauses, except for 88 level entries.

(8) The description of data-name-1, or of any item subordinate to it, must not contain an OCCURS clause with the DEPENDING option.

(9) Data-name-1 must not be subordinate to an item containing an OCCURS clause.

(11) An item subordinate to data-name-2, must not be an OCCURS clause with the DEPENDING option.

(12) Between data-name-1 and data-name-2, there must not be any entries having a lower level number than data-name-1 and data-name-2.

(13) The length of data-name-1, multiplied by the number of its occurences, must be equal to data-name-2, except when the REDEFINES clause appears in a level 01 entry.

(14) Data-name-2 must not be qualified.

```
e.g.  02   VALUE-N       PICTURE  9(6)V99.
      02   VALUE-X REDEFINES VALUE-N PICTURE X(8).
```

Thus the above area of store can contain data in numeric format (6 digits + 2 decimal places) identified by the data-name VALUE-N, or as an 8-character alphanumeric field, identified by the name VALUE-X.

5.3.7 OCCURS CLAUSE

The OCCURS clause is used in defining related sets of repeated data, such as tables, lists, vectors, matrices, etc..

The format for the OCCURS clause is

OCCURS integer TIMES [DEPENDING ON data-name]

The clause specifies the number of times that a data item is repeated. Record Description clauses apply to each repetition of the item. When the OCCURS clause is used, the data-name that is the defining name of the entry must be subscripted when it appears in the Procedure Division. If this data-name is subscripted when they are used.

The OCCURS clause must not be used in any Data Division entry that has a level number 01, 77 or 88.

The OCCURS clause is considered to have two options.

Simple

OCCURS integer TIMES

Depending

OCCURS integer TIMES DEPENDING ON

In the simple option, integer is the exact number of occurences, but in the depending option, integer is the maximum number of occurences.

For both options note that the number of character positions occupied by a group item, that is described by an OCCURS clause, must be a multiple of four if it has subordinate to it an item which is computational. The key word FILLER may be used to fulfill this requirement.

For the depending option only, note the following:

(1) Only one such clause per record is allowed.

(2) The clause must appear in the description of either a group that

contains the last elementary item of the record, or, in the description of the last elementary item.

(3) The item must not be contained in a group that has either option of the OCCURS clause.

e.g.

a Data Division entry such as

```
01  Q.

    02  Q-1         PICTURE  9(4).
    02  Q-2         PICTURE  9(4).
    02  Q-3         PICTURE  9(4).
     ¦   ¦             ¦       ¦
     ¦   ¦             ¦       ¦
     ¦   ¦             ¦       ¦
     ¦   ¦             ¦       ¦
    02  Q-12        PICTURE  9(4).
```

could be written more simply as

```
01  Q.

    02  Q-1         PICTURE  9(4)  OCCURS 12 TIMES.
```

5.3.7.1

Subscription

Subscription provides for reference to data items in a table or list, that have been assigned individual data-names. If an item has an OCCURS clause, or belongs to a group that has an OCCURS clause, it must be subscripted whenever it is used.

A subscript is a positive non-zero integer, the value of which determines the element that is referred to within a table or list. The subscript may be either a literal or a data-name and is enclosed in parentheses after the terminal space of the name of the table or list. A subscript may be a display item, but should be a computational item to shorten program execution time.

Tables may be defined so that more than one level of subscription is required to locate an element. This happens when a group item is described with an OCCURS clause and contains an item described by an OCCURS clause. A maximum of three levels of subscription is permitted in COBOL. Multi-level subscripts must be written from left to right, in decreasing inclusiveness of the groupings in the table. Subscripts are written within a single pair of parentheses and are separated by a comma and space.

Subscription Example

DATA DESCRIPTION

LETTER-RACK

SHELF (1)	PIGEON-HOLE (1,1)	PIGEON-HOLE (1,2)	PIGEON-HOLE (1,3)
SHELF (2)	PIGEON-HOLE (2,1)	PIGEON-HOLE (2,2)	PIGEON-HOLE (2,3)

PART OF PROCEDURE DIVISION

MOVE LETTER TO PIGEON-HOLE (2,3).

A subscripted data-name may not be used for any of the following functions:

(1) a subscript

(2) a qualifier

(3) the defining name of a record description entry

(4) data-name-2 in a REDEFINES clause, and

(5) data-name in the DEPENDING option of the OCCURS clause.

e.g. The Data Division entry:-

```
01   LETTER-RACK.
     02   SHELF OCCURS 2 TIMES.
          03   PIGEON-HOLE  PICTURE X OCCURS 3 TIMES.
```

will set up a 2×3 table.

5.4 EXAMPLES OF RECORD DESCRIPTION ENTRIES

5.4.1 GROUP ITEM FORMAT

The most common form of Group Item format is

```
01   TIME-CARD.
```

TIME-CARD in this case would be called a RECORD-NAME. The record would be further defined with higher level numbers.

The following addition may be made to the Group Item format:

a) 01 TIME-CARD USAGE IS COMPUTATIONAL. (see 5.2.6.6)

b) 01 DAY-CARD REDEFINES TIME-CARD.

This type of entry may only be used in the WORKING-STORAGE SECTION. (Redefining at 01 levels in the FILE SECTION is catered for by the DATA RECORDS clause.)

The clause states that record-name DAY-CARD occupies the same area in store as record-name TIME-CARD.

c) 02 HOUR-CARD OCCURS 12 TIMES. (see 5.3.7)

[03 DAY-CARD OCCURS 5 TIMES PICTURE 99.]

The 02 level description would be complemented by an elementary item (see 5.4.2).

d) 01 DAY-CARD USAGE IS COMPUTATIONAL.

This clause specifies that all items further described at an elementary level are COMPUTATIONAL.

5.4.2 ELEMENTARY ITEM FORMAT

```
a)      02   CARD-ITEM    PICTURE X(4).
        02   CARD-ITEM    PICTURE XXXX.
        02   CARD-ITEM    PICTURE 9(4).
        02   CARD-ITEM    PICTURE 9V999.
```

These four examples show some of the possibilities of describing a four character field.

```
b)      77   PAGE-COUNT   PICTURE 9(4) USAGE IS COMPUTATIONAL
                                       VALUE IS 0.
```

This clause describes a field PAGE-COUNT as being a 4 character numeric field, whose initial value is 0, and is held in COMPUTATIONAL format.

```
c)      02   C-DATE       PICTURE 9(6).
        02   D-DATE REDEFINES C-DATE.

             03   C-DAY    PICTURE 99.
             03   C-MTH    PICTURE 99.
             03   C-YR     PICTURE 99.
```

This example of the REDEFINES statement shows how it is possible to describe the date (1st Sept. 1971 ≡ 010971) in two ways, so that the programmer may be able to extract the value of the month say, from a field containing the date.

```
d)      02   HOUR-CARD OCCURS 12 TIMES.

             03   DAY-CARD OCCURS 5 TIMES PICTURE 99.
```

These statements set up a 12×5 array in store.

Data would be moved into the array with the following PROCEDURE DIVISION entry

```
MOVE D-VALUE TO DAY-CARD (X, Y)
```

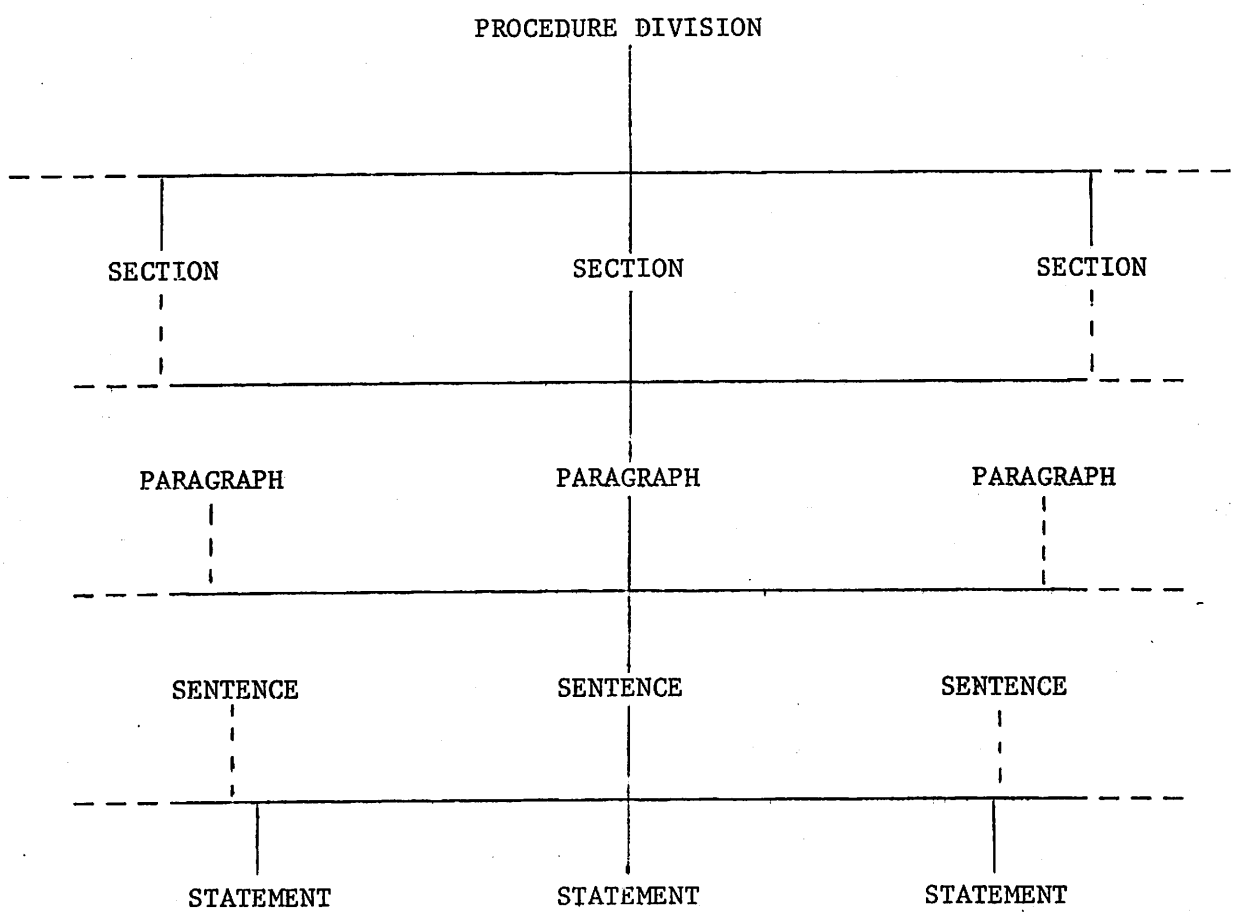
where X & Y are either numeric values or data-names indicating the element of the array to which the data is being moved.

6.0 PROCEDURE DIVISION

The Procedure Division of a source program specifies those processing steps required for the input, output and manipulation of data. These steps are expressed in a restricted form of English, which employs verbs to denote specific actions, with statements and sentences to describe complete procedures. This division must begin with the words PROCEDURE DIVISION, followed by a full-stop and space. The last statement of the PROCEDURE Division ends the source program.

6.1 SYNTAX

The following diagram illustrates the general syntactic structure of the Procedure Division.



6.1.1 STATEMENTS

A statement consists of a COBOL verb, or the word IF or ON, followed by any appropriate operands (data-names, file-names, or literals) and any other COBOL words that are necessary. The three types of statements are compiler-directing, imperative and conditional.

COMPILER-DIRECTING STATEMENT :

A compiler-directing statement instructs the compiler to take certain actions at compilation time; it contains one of the verbs ENTER, EXIT, NOTE and its operands. ENTER and EXIT statements must appear as separate sentences.

IMPERATIVE STATEMENT :

An imperative statement specifies an unconditional action to be taken by the object program; it consists of a COBOL imperative verb and its operands.

CONDITIONAL STATEMENT :

A conditional statement tests a specified condition to determine which path in a program is to be taken.

6.1.2 SENTENCES

A sentence is a single statement, or a series of statements, terminated by a fullstop and space. The separator between statements is space(s), which may be preceeded by a comma, a semicolon or space(s) or THEN. A sentence must begin in or after margin B.

6.1.3 PARAGRAPHS

A paragraph is a logical entity comprising one or more sentences.

Each paragraph must begin with a paragraph-name. The format of a paragraph is

PARAGRAPH-NAME.

sentences .....

Sentences may start on the same line as paragraph-name.

Paragraph-names are procedure-names and follow the rules of word formation. They may be composed solely of numeric characters but, unless they are identical, they are not equivalent, e.g. 0023 is not equivalent to 23.

A paragraph-name must not be ambiguous. All paragraph-names and section-names must be unique except when it is possible to ensure uniqueness by qualifying a paragraph-name with IN or OF followed by a section-name.

Qualification need not be employed in reference to a paragraph from within the same section. A paragraph ends at the next paragraph-name or section-name, or at the end of the Procedure Division.

A paragraph must not contain statements to which control would not flow.

6.1.4 SECTIONS

A section is composed of one or more successive paragraphs and must begin with a section-header. A section header has the following format

section-name SECTION [integer].

A section-header must be terminated by a fullstop and be on a line by itself.

Section-name must conform to the rules of procedure-name formation, as for paragraph names. A section-header need not immediately follow the words PROCEDURE DIVISION.

Integer is the priority-number.

6.2 ARITHMETIC EXPRESSIONS

Certain statements in the Procedure Division may have operands that are arithmetic expressions. An arithmetic expression comprises arithmetic operators, data-names, and/or literals representing items, on which arithmetic may be performed.

The following five arithmetic operators may be used in arithmetic expressions.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Parentheses may be used to indicate the order of operations.

ARITHMETIC EXPRESSIONS are used with the COMPUTE statement (see 6.6.1).

6.3 INPUT AND OUTPUT STATEMENTS

6.3.1 OPEN STATEMENT

The OPEN statement primes the system for the processing of files. Where applicable, execution of an OPEN statement causes labels to be checked; for output files, labels are created.

The format of the OPEN statement is

$$\underline{\text{OPEN}} \quad \left\{\begin{matrix}\text{INPUT-clause} \\ \text{OUTPUT-clause}\end{matrix}\right\} \quad , \quad \left\{\begin{matrix}\text{OUTPUT-clause} \\ \text{INPUT-clause}\end{matrix}\right\}$$

The format of the INPUT clause is

OPEN INPUT filename [WITH NO REWIND].

The format of the OUTPUT clause is

OPEN OUTPUT filename [WITH NO REWIND].

e.g. OPEN INPUT CARDFILE, OUTPUT MASTER-FILE, PRINTOUT.

The OPEN statement must be executed prior to any other input or output statement for a file. A second OPEN statement for a given file cannot be executed before the execution of a CLOSE statement for that file.

The OPEN statement does not make an input record available for processing. A READ statement must be executed to obtain the first record.

6.3.2 READ STATEMENT

The READ statement makes the next logical record available from an input file and allows the execution of specified imperative statements when the end of the file is detected.

The format of the READ statement is

READ file-name AT END imperative statement.

e.g. READ CARDFILE AT END GO TO FINIS.

When a READ statement is executed, the next logical record in the named file becomes accessible in the input area, as defined by the associated Record Description entry. The file-name must be defined by a File Description entry in the Data Division.

The record remains available in the input area until the next READ, or CLOSE, statement for that file is executed.

If more than one Record Description is given following the FD-entry for the file, it is the programmer's responsibility to recognise which description applies.

If, on execution of a READ statement, the end of the file is detected, the imperative statements in the AT END clause are executed. Note that, after the AT END branch has been taken, no input record is available.

After the AT END branch has been taken for a file, any attempt to read from that file constitutes an error, unless the file has been closed and re-opened.

For magnetic tape files, the READ statement causes the following operations to take place, when the end of a volume is reached (multi-reel file).

(1) A volume switch is made.

(2) The volume header label-checking procedure is executed.

(3) The next logical record in the file is made available.

6.3.3 WRITE STATEMENT

The WRITE statement outputs a logical record to an output file.

Format of the WRITE statement is

WRITE record-name [BEFORE/AFTER ADVANCING integer LINES]

The AFTER/BEFORE ADVANCING option is only used for output to a lineprinter.

e.g. WRITE PRINT-REC AFTER ADVANCING 1 [LINES].

An OPEN statement must be executed before the execution of the first WRITE statement for a file. The logical record, referred to by record-name, is not available after a WRITE statement has been executed.

For magnetic tape files, the WRITE statement causes the following operations to take place, when the end of a volume is reached:

(1) An end-of-volume label is written

(2) A volume switch is made.

(3) The header label-writing procedure is executed.

(4) The current logical record is output.

The AFTER/BEFORE ADVANCING option is used to enable the programmer either to move the paper a number of lines, or to perform a recognition throw, after or before a line is printed. If this option is written for a given file, every WRITE statement for the file must also contain this option.

Records in a printer file must start with a one word item, i.e. 4 characters if DISPLAY, 6 digits if COMPUTATIONAL. If the AFTER/BEFORE ADVANCING option of the write statement is used, this item if filled with the appropriate value. Hence, it is normally defined as FILLER. If this option is not used, the user must fill the item with the paper movement throw value.

6.3.3.1

New Page

A new page on the lineprinter is obtained by using the WRITE Statement in the following way :

WRITE recordname AFTER/BEFORE ADVANCING data-name.

where data-name is defined in the WORKING-STORAGE SECTION of the DATA DIVISION as :

```
77 dataname      PICTURE X VALUE '2' DISPLAY.
```

6.3.4 CLOSE STATEMENT

The CLOSE statement terminates the processing of a file or a reel of a file.

The format of the CLOSE statement is

CLOSE filename [REEL/UNIT WITH NO REWIND/LOCK]

When a CLOSE statement is executed, the current unit of the file is closed. The CLOSE statement may only be specified for a file that is open. After a CLOSE statement with the REEL or UNIT option has been executed, an OPEN statement is not required before further reference is made to the file. For any other CLOSE statement, OPEN statement must be executed before any reference can be made to that file.

A CLOSE statement with any of the options given may only be written for files assigned to magnetic tape.

(1) If the WITH LOCK option is specified, the reel is rewound and set to manual.

(2) If the WITH NO REWIND option is specified, the file is closed but no subsequent rewind occurs.

(3) If neither the WITH LOCK nor the WITH NO REWIND option is specified the reel is rewound but not set into manual.

(4) If the REEL or the UNIT option (REEL and UNIT are synonymous) is specified, the volume-switching procedures are carried out.

e.g. CLOSE CARDFILE, MASTER-FILE, PRINTOUT.

6.3.5 DISPLAY STATEMENT

The DISPLAY statement displays data on the console. The format of the DISPLAY statement:

DISPLAY data-name/literal.

e.g. DISPLAY 'TYPE IN DATE OF RUN '.

6.3.6 ACCEPT STATEMENT

The ACCEPT statement obtains data from the console.

The format of the ACCEPT statement is

ACCEPT data-name

e.g. ACCEPT DATE-IN.

Data-name may not be a variable length item, nor may it be a numeric item. One logical record is read and the appropriate number of characters is moved into data-name. There is no editing or error-checking of the incoming data.

6.4 DATA MANIPULATION STATEMENTS

6.4.1 MOVE STATEMENT

The MOVE statement moves data from one area of core store to another and is able to perform conversions and editing on the data that is moved.

The MOVE statement has the following formats

```
                    data-name-1
MOVE                literal              TO data-name-2,   [data-name-3,...]
                    figurative-constant

MOVE CORRESPONDING  data-name-1          TO data-name-2.
```

EXAMPLES OF SIMPLE MOVE STATEMENTS:

```
MOVE DATE-IN TO DATE-OUT.
MOVE 25 TO LINE-COUNT.
MOVE ZEROS TO TOT-1  A-TOT-1  TOT-2  SUB-1  VALUE-1  PRICE-1.
```

When the simple move statement is used, the data contained in data-name-1, or specified by literal, is moved to data-name-2. This information is also moved to any other receiving areas mentioned in the statement.

When a group item is involved in a simple move, the data is moved without regard to the level structure, without editing.

The following considerations affect elementary item movement.

(1) Numeric to numeric movement

The items are aligned by decimal points, with insertion of zeros or truncation at either end, as required.

Conversion to the USAGE of the receiving field takes place when the USAGE clauses of the source and receiving fields differ.

(2) All other permissible movements.

The characters are placed in the receiving area from left to right, unless the receiving field is specified as JUSTIFIED RIGHT.

The remaining positions are filled with spaces if the receiving field is not completely filled by the data being moved.

EXAMPLES OF DATA MOVEMENT

- Source Field -		- Receiving Field -		
PICTURE	Value	PICTURE	Value before MOVE	Value after MOVE
99V99	1234	99V99	9876	1234
99V99	1234	99V9	987	123
9V9	12	99V999	98765	01200
XXX	A2B	XXXXX	Y9X8W	A2B
9V99	123	99.99	87.65	01.23
AAAAAA	REPORT	AAA	JKL	REP

6.4.2 MOVE CORRESPONDING STATEMENT

MOVE CORRESPONDING data-name-1 TO data-name-2

When the move corresponding statement is used, selected items within data-name-1 are moved to selected areas within data-name-2. The effect is as though a series of simple MOVE statements had been written to move data between fields that have the same name, within the group items data-name-1 and data-name-2. The rules stated for the simple MOVE apply to each pair of corresponding items.

The following rules apply to the CORRESPONDING option.

(1) At least one of two corresponding items must be elementary.

(2) Items are corresponding data items if the respective data-names, including all qualification up to but not including data-name-1 and data-name-2, are the same.

(3) Data-name-1 and data-name-2 must be group items.

(4) A data-item, that is subordinate to data-name-1 or data-name-2, and that contains a REDEFINES or OCCURS clause in its description, is ignored. However, data-name-1 and data-name-2 may be described with REDEFINES or OCCURS clauses, or may be subordinated to items described with REDEFINES or OCCURS clauses.

(5) If either data-name-1 or data-name-2 is described with an OCCURS clause, it must be subscripted.

(6) In evaluating correspondence, only the first description of an area that has been redefined is used. Consider the following data organisation.

```
01  A.

    02  B                   PICTURE X(4).
    02  C REDEFINES B.

        03  D                  PICTURE X.
        03  E                  PICTURE X(3).

    02  F                      PICTURE 9(3).
```

If A is data-name-1 or data-name-2, only B or F can be considered as potential corresponding items.

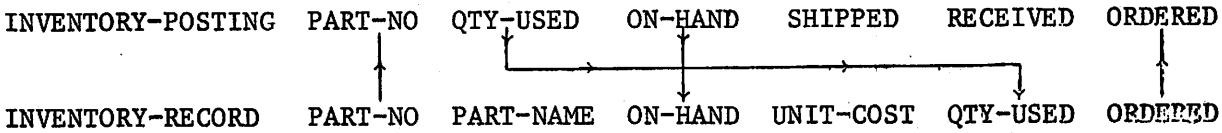
(7) Neither data-name-1 nor data-name-2 may be data items with level numbers 77 or 88.

To illustrate the use of MOVE CORRESPONDING, suppose that the programmer wishes to transfer corresponding items from a work area named INVENTORY-POSTING to an output area designated INVENTORY-RECORD. He could write the statement

```
MOVE CORRESPONDING INVENTORY-POSTING TO INVENTORY-RECORD
```

The example shows the movement of data that might result from this statement. Note that non-corresponding items in the receiving area are not affected.

MOVE CORRESPONDING EXAMPLE



DATA DESCRIPTIONS

```
01   INVENTORY-POSTING.

     02   PART-NO          PICTURE  999.
     02   QTY-USED         PICTURE  9(4).
     02   ON-HAND          PICTURE  9(4).
     02   SHIPPED          PICTURE  999.
     02   RECEIVED         PICTURE  9(4).
     02   ORDERED          PICTURE  9(4).

01   INVENTORY-RECORD.

     02   PART-NO          PICTURE  999.
     02   PART-NAME        PICTURE  X(15).
     02   ON-HAND          PICTURE  9(4).
     02   UNIT-COST        PICTURE  999V99.
     02   QTY-USED         PICTURE  9(4).
     02   ORDERED          PICTURE  9(5).
```

6.4.3 EXAMINE STATEMENT

The EXAMINE statement replaces certain occurrences of a given character, or counts the number of such occurrences in a data item, or both.

The EXAMINE statement has the following formats

EXAMINE data-name TALLYING { ALL / LEADING / UNTIL FIRST / FIRST } 'character-1' [REPLACING BY 'character-2']

EXAMINE data-name REPLACING { ALL / LEADING / UNTIL FIRST / FIRST } 'character-1' BY 'character-2'

data-name must refer to a display item.

The use of figurative constants instead of character-1 or character-2 is permitted. Character-1 and character-2 must be single-character, non-numeric literals (i.e. enclosed in quotation marks) and must be members of the set of allowable characters for the data item. For example, a '2' can not replace an 'A' in an alphabetic item.

Examination always proceeds from left to right.

TALLYING OPTION

When the TALLYING option is used, a count is made at run time of the number of occurrences of the specified character in data-name. This count replaces the value of the data item TALLY. TALLY is assumed, by the compiler, to exist with the following data description; it must not be declared in the program.

```
77 TALLY PICTURE IS 9 (5), USAGE IS COMPUTATIONAL .
```

TALLY may also be used as a data-name in other procedural statements wherever the format permits.

The value in TALLY depends upon which of the following is used.

(1) If ALL is specified, all occurrences of character-1 in the data item are counted.

(2) If LEADING is specified, the count gives the number of occurrences of character-1, prior to encountering a character other than character-1.

(3) If UNTIL FIRST is specified, the count gives the number of characters encountered prior to the first occurrence of character-1.

BOTH OPTIONS

When the REPLACING Option is used, the replacement of characters depends upon which of the following is used.

(1) If ALL is specified, character-2 is substituted for each occurrence of character-1.

(2) If LEADING is specified, character-2 is substituted for character-1 until a character other than character-1, or the right-hand boundary, is encountered.

(3) If UNTIL FIRST is specified, character-2 is substituted for each character until first character-1, or the right-hand boundary, is encountered.

(4) If FIRST is specified, only the first character-1 is replaced by character-2.

EXAMPLES OF DATA EXAMINATION:

Statement	ITEM-1 Before	ITEM-1 After	Value of Tally
EXAMINE ITEM-1 TALLYING ALL '0'	101010	101010	3
EXAMINE ITEM-1 TALLYING ALL '1' REPLACING BY '0'	101010	000000	3
EXAMINE ITEM-1 REPLACING LEADING '*' BY SPACE	**700*	700*	unchanged
EXAMINE ITEM-1 REPLACING FIRST '*' BY '£'	**1	£*1	unchanged

6.4.4 TRANSFORM Statement

The TRANSFORM statement alters characters in a data item according to a programmer specified transformation rule.

The format of the TRANSFORM statement is

TRANSFORM data-name-1 CHARACTERS

FROM { data-name-2 / non-numeric-literal-1 / figurative-constant-1 }

TO { data-name-3 / non-numeric-literal-2 / figurative-constant-2 }

Data-name-1 must be an elementary alphabetic, alphanumeric, or report item, or must be a group item.

6.5 ENDING STATEMENT

6.5.1 STOP Statement

The STOP statement terminates or delays the continued execution of the object program.

The format of the STOP statement is

STOP { RUN / literal }

The STOP RUN statement terminates execution of the object program and returns control to the system execution.

The STOP literal statement causes the specified literal to be typed on the console and the object program to pause, awaiting operator intervention. A fullstop must be typed in to resume execution. The literal is restricted to a maximum of 72 characters.

e.g. STOP SET UP INVOICE STATIONERY

6.6 ARITHMETIC STATEMENTS

All data names used in arithmetic statements must represent elementary numeric data items, except when the items are the operands of the GIVING or CORRESPONDING options.

The maximum size of any data item or literal is 12 decimal digits.

Decimal point alignment is supplied automatically throughout the computations.

6.6.1 COMPUTE Statements

The COMPUTE statements assigns to a data item the value of a numeric data item, literal, or arithmetic expression.

The format of the COMPUTE statement is

```
                                                    numeric-literal
COMPUTE data-name-1 [ROUNDED]  =                    data-name-2
                                                    arithmetic-expression
                        [ON SIZE ERROR imperative-statement]
```

e.g.
```
COMPUTE MINUTES ROUNDED = (HOURS * 60) + MINS + (SECONDS / 60)
                ON SIZE ERROR PERFORM ERROR-ROUTINE.
```

where MINUTES has a picture 999V9.

If HOURS has a value of 10, MINS has a value of 25 and SECONDS has a value of 15, the above statement would place the value 625.3 in the field MINUTES.
[Note that if the ROUNDED option had not been specified the result would be 625.2 (see ROUNDED OPTION page 70)].
If, in the above statement, HOURS has a value of 100, the ON SIZE ERROR option is executed, because the result, to be placed in MINUTES, is too great to be held in that field (see SIZE ERROR OPTION page 71).
Thus paragraph ERROR-ROUTINE would be executed and control returned to the next sentence.

If, with field HOURS having a value of 100 as above, the ON SIZE ERROR option is not specified, the program flow continues without interruption, but no assumption should be made as to the contents of field MINUTES.

6.6.2 ADD Statement

The ADD statement adds two or more numeric values.

The ADD statement has the following formats

```
ADD     numeric-literal  ..... TO       data-name-n
        data-name-1              GIVING
[ROUNDED ON SIZE ERROR imperative-statement .....]
ADD CORRESPONDING data-name-1 TO data-name-2
[ROUNDED ON SIZE ERROR imperative statement .....]
```

When the TO option is used the values of all the data-names (including data-name-n) and literals in the statement are added and the resulting sum is placed in data-name-n. When the GIVING option is written, at least two data-names and/or numeric literals must follow the word ADD and the initial value of data-name-n is excluded from the addition.

e.g. If field A has a value of 120, B=80, C=50 and D=75.

ADD A, B, C, D GIVING E.

will give E=325 with A,B,C and D unchanged.

ADD A, B, C TO D.

will give D=325 leaving A,B and C unchanged.

The CORRESPONDING option of the ADD statement allows the programmer to specify the addition of corresponding data items in one operation. This is analogous to a MOVE statement with the CORRESPONDING option. Data-name-1 and data-name-2 must be group items. The rules stated for arithmetic statements apply to each pair of elementary items.

Rules 2 to 6 listed for the MOVE CORRESPONDING statement also apply to the ADD CORRESPONDING statement.

When the ON SIZE ERROR option is used with CORRESPONDING the error branch is taken only after the completion of all the ADD operations. If any of the additions produce a size error condition, the resultant field for that addition would remain unchanged and the imperative statements specified in the ON SIZE ERROR clause would be executed.

When the ROUNDED option is used with CORRESPONDING, the result of each ADD operation is rounded.

6.6.3 SUBTRACT STATEMENT

The SUBTRACT statement subtracts one numeric data item, or a sum of several numeric data items, from a specified item.

The SUBTRACT statement has the following formats

```
SUBTRACT    data-name-1
            numeric-literal-1 .....................
FROM        data-name-m          GIVING data-name-n
            numeric-literal-m    GIVING data-name-n
[ROUNDED ON SIZE ERROR imperative statement .....]
 SUBTRACT CORRESPONDING data-name-1 FROM data-name-2
[ROUNDED on SIZE ERROR imperative statement .....]
```

The SUBTRACT statement adds the values of all the operands that procedes FROM, then subtracts the sum from the value of the item that follows FROM. A literal may follow FROM only when the GIVING option is specified.

The CORRESPONDING option of the SUBTRACT STATEMENT is analogous to the CORRESPONDING option of the ADD statement.

e.g. with values for fields A and B as before

SUBTRACT B FROM A GIVING C.

will give C=40 with A and B unchanged.

SUBTRACT B FROM A.

gives A=40 with B unchanged.

SUBTRACT A,B,C FROM 300 GIVING D.

will give D=50 with A, B and C unchanged.

6.6.4 MULTIPLY STATEMENT

The MULTIPLY statement multiples two numeric data items.

The MULTIPLY statement has the following formats:-

```
MULTIPLY  data-name-1          BY  data-name-2
          numeric-literal-1        numeric-literal-2

    [GIVING data-name-3] [ROUNDED] [ON SIZE ERROR imperative statement]
```

The MULTIPLY statement multiplies the first item by the second item, putting the result in data-name-2, or data-name-3 if the GIVING option is written. A literal may only follow BY if the GIVING option is specified.

For field A=120, B=80

```
MULTIPLY A BY B.
```

gives B=9600 and A is unchanged.

```
MULTIPLY A BY 3.5 GIVING B.
```

gives B=420, leaving A unchanged.

6.6.5 DIVIDE Statement

The DIVIDE statement divides one numeric data item by another.

The format of a divide statement is

```
DIVIDE    data-name-1
          numeric-literal-1
INTO      data-name-2          [GIVING data-name-3]
          numeric-literal-2     GIVING data-name-3
[ROUNDED] [ON SIZE ERROR imperative statement .....]
```

The DIVIDE statement divides the second item by the first item, putting the result in data-name-2, or data-name-3 if the GIVING option is specified.

Division by zero results in a SIZE ERROR condition.

Where field A has a value of 120, and B=20.

DIVIDE 15 INTO A.

gives A=8.

DIVIDE B INTO A GIVING C.

gives C=6 with A and B unchanged.

6.6.6 OPTIONS

The ROUNDED and SIZE ERROR options apply to all of the arithmetic statement. The GIVING option applies to all arithmetic statements except COMPUTE.

GIVING OPTION

If the GIVING option is written, the result of the calculation is placed in the item that follows GIVING. This item may be a report item.

ROUNDED OPTION

Truncation occurs (unless the ROUNDED option has been specified), if the number of places calculated for the result, after decimal point alignment, is greater than the number of places in the data item that is to receive the result.

When the ROUNDED option is specified, the least significant digit of the result is increased by 1, if the next digit is greater than or equal to 5. A negative result is rounded by rounding its modulus and negating it.

	- Item to Receive Calculated Results -		
Calculated Results	PICTURE	Value After Rounding	Value After Truncating
- 6.42	S9V9	- 6.4	- 6.4
-12.36	S99V9	-12.4	-12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	99V	66	65
.0055	V999	.006	.005

SIZE ERROR Option

If truncation would have occurred at the left hand end of a receiving field, a size error occurs.

If the SIZE ERROR option has been specified and a size error condition arises, the contents of the receiving field are not altered and the imperative statement following SIZE ERROR are executed.

If the SIZE ERROR option has not been specified and a size error condition arises, no assumption should be made about the result. The program flow is not interrupted.

Note that the size error condition applies to both final and intermediate results.

An arithmetic statement, if written with a SIZE ERROR option is not an imperative statement. Rather, it is a conditional statement and is prohibited in contexts where only imperative statements are permitted.

6.7 CONDITIONAL STATEMENTS

A conditional statement is a means of altering the flow of a program according to the conditions prevailing at the time of execution of the statement. For example, the condition may be a particular value of a data item or the status of a sense switch.

6.7.1 Evaluation of Conditional Statements

When a condition is evaluated the following action is taken.

(1) If the value of the condition is true, the statements immediately following the condition are executed.

(2) If the value of the condition is false and the conditional statement is an IF or ON statement, the statements following ELSE or OTHERWISE are executed. If the ELSE/OTHERWISE option is omitted, the next sentence is executed.

(3) If the value of the condition is false and the conditional statement is not an IF or ON statement, the next sentence is executed.

The AT END, and SIZE ERROR conditional statements are followed by a series of imperative statements.

In a series of imperative statements, executed if a condition is true, only the last statement may be an unconditional GO TO statement or a STOP RUN statement. Otherwise, the series of statements would contain statements to which control cannot flow. In the following example, the statement MOVE A to B could never be executed whether the AT END condition were found to be true or false.

```
READ CURRENT-INVOICE
AT END MOVE CUST-NO TO LAST-CUST
GO TO COPY-MAIN-FILE
MOVE A TO B.
```

6.7.2 TEST-CONDITIONS

A test-condition is an expression that may be either true or false, depending upon the circumstances existing when the expression is evaluated.

There are four types of simple test-conditions:

relation test

sign test

class test

condition-name test.

The word NOT may be used in any simple test-condition to reverse its meaning. For example, AGE NOT GREATER THAN 21 is the opposite of AGE GREATER THAN 21. NOT may also precede an entire condition, as in NOT (AGE GREATER THAN 21) which has the same meaning as AGE NOT GREATER THAN 21.

In a test-condition a data-name may be qualified but not subscripted.

6.7.2.1 RELATION Test

A relation test involves the comparison of two operands, either of which may be a data-name, a literal, or an arithmetic expression.

data-name-1 arithmetic-expression-1 figurative-constant-1 literal-1	IS NOT	> < = GREATER THAN LESS THAN EQUAL TO	data-name-2 arithmetic-expression-2 figurative-constant-2 literal-2

The comparison of two literals is not permitted. A figurative constant may be used instead of either literal in a relation test.

The symbol > is equivalent to the reserved words GREATER THAN. The

symbol < is equivalent to the reserved words LESS THAN. The symbol = is equivalent to the reserved words EQUAL TO.

Comparison of Numeric Items

For numeric items, a relation test determines that the value of one of the items is less than, equal to, or greater than the other, regardless of the field length. Numeric items are compared algebraically, after alignment of their decimal point.

Comparison of Non-numeric Items

For non-numeric items, a comparison results in the determination that one of the items is less than, equal to, or greater than the other, with respect to the binary collating sequence of characters in 4100 internal code.

If the two non-numeric items are of the same length, their characters in corresponding positions are compared from left to right until either a pair of unequal characters, or the end of the items is encountered. When two unequal characters, or the end of the items is encountered, the item containing the character that has a higher binary value is the greater. The items are considered to be equal after the last pair of characters is found to be equal.

If the two non-numeric items are of unequal length, the shorter is considered to be padded with spaces to the right, so that the items are of the same length. Comparison then proceeds as for items of equal length.

6.7.2.2

SIGN Test

The sign test determines whether the value of a numeric item is negative, zero or positive.

The format for a sign test is

```
data-name                                  POSITIVE
arithmetic-expression    IS [NOT]          ZERO
                                           NEGATIVE
```

Note that the value zero is considered to be neither positive nor negative.

6.7.2.3

CLASS Test

A class test examines an item to determine its composition.

The format for a class test is

```
data-name    IS [NOT]    NUMERIC
                         ALPHABETIC
```

The two homogeneous classes are

NUMERIC which contains only the characters 0 to 9

ALPHABETIC which contains only the characters A to Z and space.

The item to be tested must be elementary alphanumeric, alphabetic, or numeric display.

6.7.2.4

CONDITION-NAME Test

The format for a condition-name test is

```
[NOT]  condition-name
```

A condition-name test is one in which a conditional variable is compared with the value specified for an associated condition-name.

For example, the following item might appear in a payroll program.

```
02   MARITAL-STATUS PICTURE 9.
88   SINGLE VALUE IS 1.
88   MARRIED VALUE IS 2.
88   DIVORCED VALUE IS 3.
88   OTHER VALUE IS 4.
```

As a shorthand form of the simple relational condition MARITAL-STATUS = 1, the programmer could write the single condition-name SINGLE. Therefore, the following two statements would produce an identical result.

```
IF MARITAL-STATUS = 1 GO TO E.
IF SINGLE GO TO E.
```

Hence, the condition-name test is an alternative form of certain conditions that could be expressed by a single relational condition.

6.7.3 COMPOUND CONDITIONS

Simple test-conditions may be combined with logical operators, according to specified rules, to form compound conditions. The logical operators are AND, OR and NOT. Two or more simple conditions combined by OR, AND or both, constitute a compound condition.

The word OR is used to mean either or both. Thus, A OR B is true when A is true, or B is true, or both A AND B are true.

The word AND is used to mean both. Thus, A AND B is true only if both A and B are true.

The word NOT is used in the manner described in 6.7.2. Thus, NOT (A OR B) is true if both A and B are false. NOT (A AND B) is true if A is false, B is false, or if both A and B are false.

Parentheses may be used to specify the order in which conditions are evaluated. Logical evaluation begins with the innermost pair of parentheses and proceeds to the outermost. If the order of evaluation is not specified by parentheses, AND and its surrounding conditions are evaluated before OR and its surrounding conditions.

Thus, the expression

A IS GREATER THAN B OR A IS EQUAL TO C AND D IS POSITIVE

would be evaluated as if it were

(A IS GREATER THAN B) OR ((A IS EQUAL TO C) AND (D IS POSITIVE)).

6.7.4 IF Statement

The format of the IF statement is

IF condition THEN {statement-1....} {ELSE} {statement-2....}
{NEXT SENTENCE} {OTHERWISE} {NEXT SENTENCE}

When the IF statement is the last clause of a sentence, ELSE (or OTHERWISE) NEXT SENTENCE may be omitted.

6.7.5 Nested IF Statements

Statement-1 and statement-2 in the IF statement may consist of one or more imperative statements and/or a conditional statement. If a conditional statement does appear, it is said to be nested.

IF statements contained within IF statements are considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered is considered to apply to the immediately preceding IF, that has not already been paired with an ELSE.

6.8 BRANCHING STATEMENTS

In the following statements, procedure-name signifies paragraph-name or section-name and may be qualified.

6.8.1 GO TO Statement

The GO TO statement transfers control from one portion of the program to another.

The GO TO statement has the following format

GO TO procedure-name-1

Procedure-name-1 in the unconditional GO TO may be omitted if the GO TO is the subject of an ALTER statement that will be executed before the GO TO statement (6.8.2). If procedure-name-1 is omitted, prior to the first execution of the GO TO statement, the program run is terminated.

6.8.2 ALTER Statement

The ALTER statement modifies an unconditional GO TO statement, thus changing the sequence in which program steps are to be executed.

The format of the ALTER statement is

ALTER {procedure-name-1 TO PROCEED TO procedure-name-2} ....

Procedure-name-1 designates a paragraph comprising a single sentence consisting only of an unconditional GO TO statement. An ALTER statement replaces the procedure-name specified in the GO TO statement with procedure-name-2, where the paragraph containing the GO TO statement is procedure-name-1. If procedure-name-1 is in a section not in store, the effect is undefined.

6.8.3 PERFORM Statement

The PERFORM statement transfers control from one part of a program to another to execute some procedure, then to return to the statement immediately following the point from which the transfer was made. The procedure may be executed a controlled number of times.

The PERFORM statement has the following formats

Simple :

PERFORM procedure-name-1 THRU procedure-name-2 [integer / date-name TIMES / UNTIL test-condition]

e.g.
```
PERFORM P-1.

PERFORM P-1 THRU P-4.

PERFORM P-1 10 TIMES.

PERFORM P-1 THRU P-6 FRED TIMES.

PERFORM P-1 THRU P-8 UNTIL SUB1 > 10.
```

Complex :

PERFORM procedure-name-1 [THRU procedure-name-2]
VARYING data-name-1 FROM [number-literal-2 / data-name-2]
BY [numeric-literal-2 / date-name-3] UNTIL test-condition-1
[AFTER date-name-4 FROM [numeric-literal-4 / data-name-5]
BY [numeric-literal-6 / data-name-6] UNTIL text-condition-2
[AFTER data-name-7 FROM [numeric-literal-8 / data-name-8]
BY [numeric-literal-9 / data-name-9] UNTIL test-condition-3]

e.g.
```
PERFORM P-1 THRU P-3
VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 10
AFTER SUB2 FROM TOM BY 1 UNTIL SUB2 = 6
AFTER SUB3 FROM DICK BY HARRY UNTIL SUB3 > FRED
```

To execute the above statement the following steps are taken:-

(i) Assign SUB1 the value 1.
Assign SUB2 the value of field TOM
Assign SUB3 the value of field DICK.

(ii) Compare SUB1 to 10. If greater, return control to the statement after the PERFORM statement.

(iii) Compare SUB2 to 6. If equal, reset SUB2 to the value of TOM, increment SUB1 by 1, go to (ii).

(iv) Compare SUB3 to field FRED. If greater, reset SUB3 to the value of field DICK, increment SUB2 by 1, go to (iii).

(v) Execute the statements contained in paragraphs P-1 through P-3. Add the value of field HARRY to SUB3. Go to (iv).

The range of a PERFORM statement comprises all those statements through which control may pass, during the execution of the PERFORM statement. The first and last statements to be executed are the first statement of procedure-name-1 and the last statement of procedure-name-2. If the THRU option is omitted, procedure-name-2 is the same as procedure-name-1.

Simple Perform Statement

The statements within the range of the PERFORM are executed once, unless the TIMES option, or the UNTIL option, is specified.

If the TIMES option is used, the statements are executed the number of times specified by integer or the initial value of data-name. Integer must have a positive value, less than 32,768. Data-name must have a PICTURE which specifies less than 7 digits and does not specify a decimal point. If the value of data-name is negative or zero, control immediately passes to the

statement following the PERFORM statement. The value of data-name is not corrupted.

If the UNTIL option is used, test-condition is examined and, if it is found to be true, the statement after the PERFORM is executed. If test-condition is false the range of the PERFORM is executed and test-condition re-examined.

Complex Perform Statement

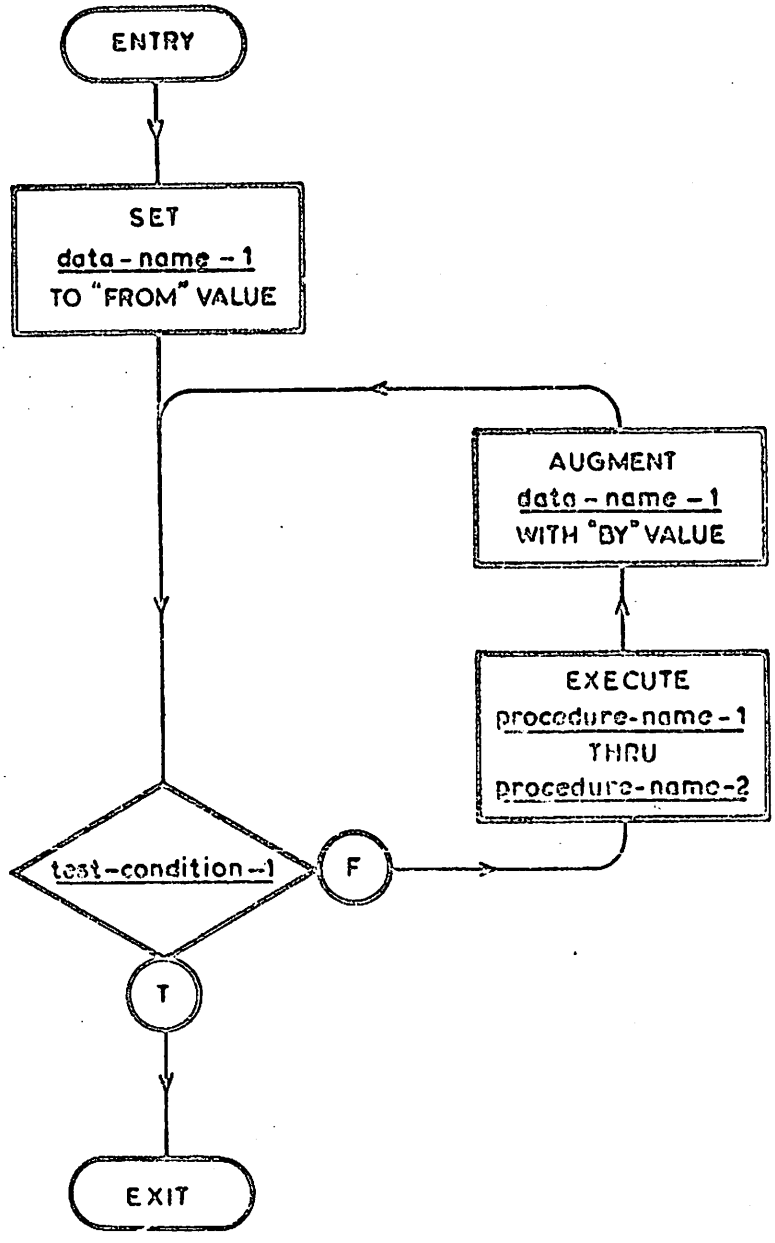
This option is used to augment the value of one or more data-names during the execution of a PERFORM statement.

When one data-name is varied, data-name-1 is set to its starting value (the FROM operand) at the start of the PERFORM statement. Test-condition-1 is then evaluated: if it is true, control passes to the statement following the PERFORM statement; if it is false, the range of the PERFORM is executed once. The value of the variant (BY) is added to data-name-1 and the condition (UNTIL) is evaluated again. The cycle continues until test-condition-1 becomes true.

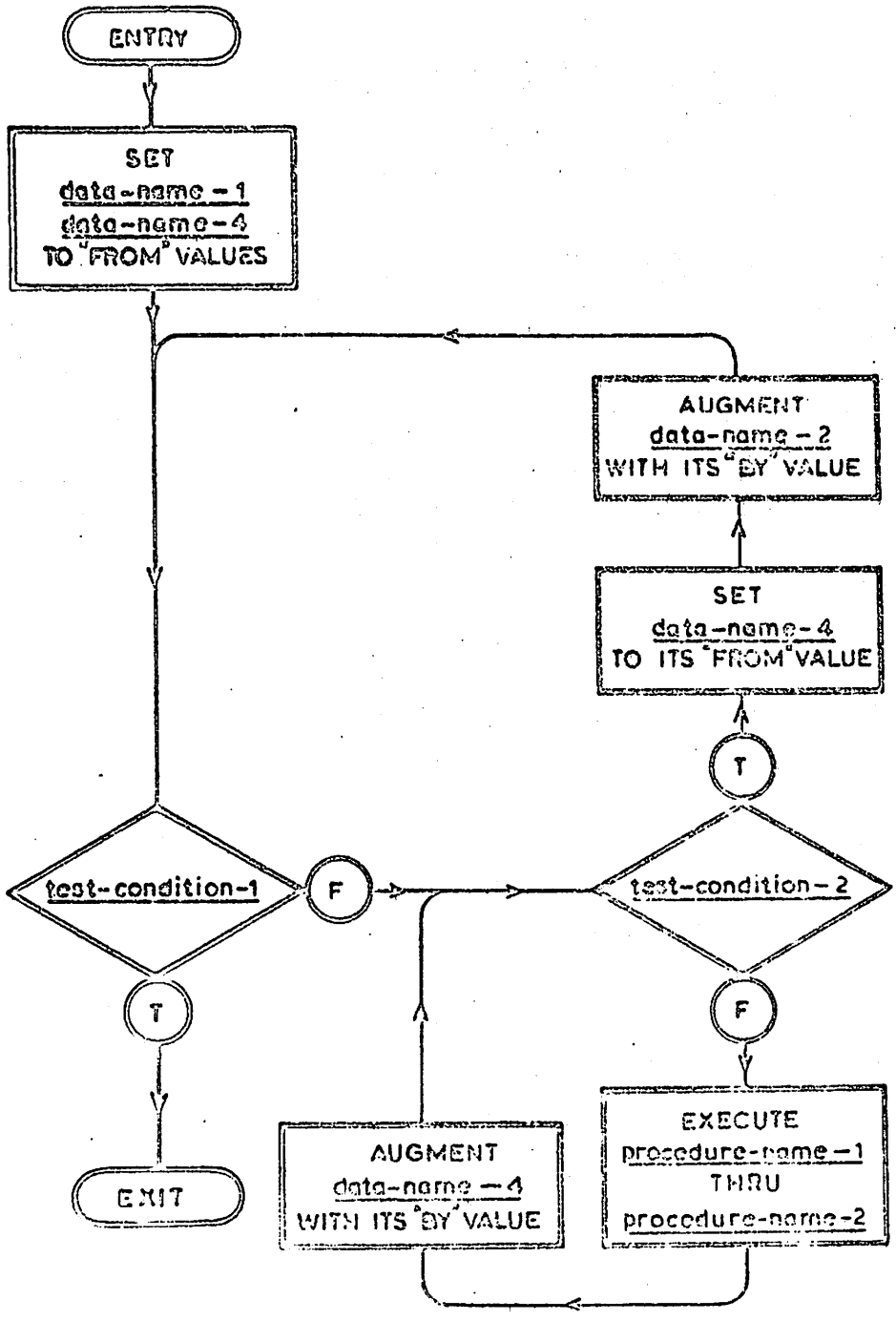
When two data-names are varied (i.e. one AFTER option), the value of data-name-4 goes through a complete cycle (FROM, BY, UNTIL) each time that data-name-1 has its BY value added. When three data-names are varied (i.e. both AFTER options), the value of data-name-7 goes through a complete cycle (FROM, BY, UNTIL) each time that data-name-4 has its BY value added, which in turn goes through a complete cycle each time data-name-1 has its BY value added.

The diagrams on the following three pages illustrate the logical flow of the complex perform. Control is transferred to the statement after the PERFORM statement when test-condition-1 is found to be true, regardless of the number of data names being varied.

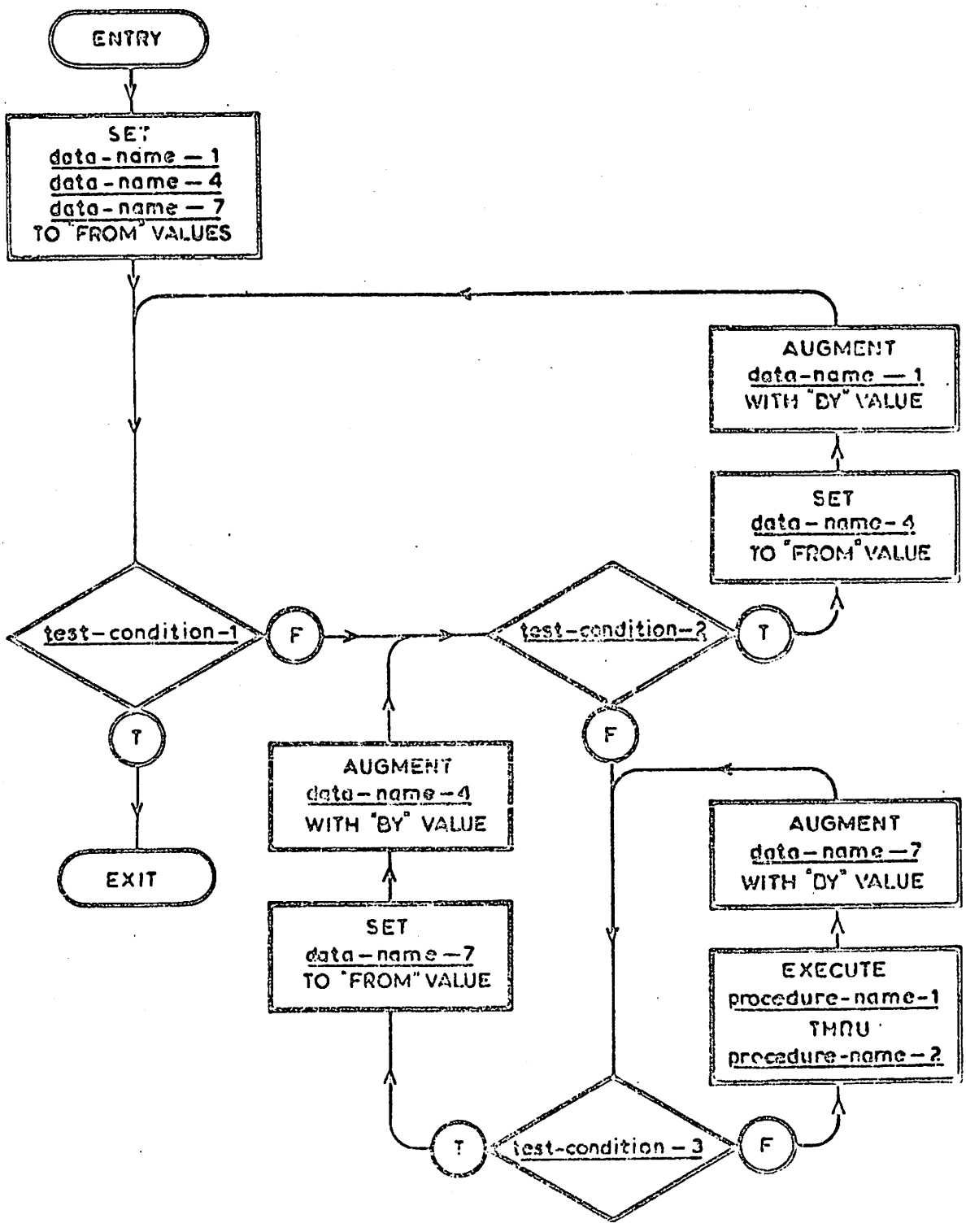
COMPLEX PERFORM FLOWCHART



COMPLEX PERFORM (ONE AFTER) FLOWCHART



COMPLEX PERFORM (TWO AFTERS) FLOWCHART



6.9 COMPILER-DIRECTING STATEMENTS

Compiler-directing statement (ENTER, EXIT and NOTE) do not cause the compiler to generate any code. They direct the compiler to adopt a particular mode of operation.

These statements must be separate sentences.

6.9.1 ENTER Statement

The ENTER statement permits the use of more than one language in a program.

The format of the ENTER statement is

ENTER language-name.

Other than COBOL, the only language that the 4100 COBOL compiler can interpret is the special purpose language, LINKAGE. LINKAGE comprises the single statement CALL.

The format of the CALL statement is

CALL subroutine-name USING parameter ....

This statement causes control to pass to the pre-compiled subroutine named subroutine-name. This subroutine is embodied in the program by the COBOL compiler from its library.

Parameter may be a non-numeric literal, numeric literal, figurative constant, data-name (which may be subscripted) or file-name. The maximum number of parameters is 15.

6.9.2 EXIT Statement

The EXIT statement provides an end point for a procedure that is to be executed by means of a PERFORM statement.

The format of the EXIT statement is

EXIT.

EXIT must appear in the source program as a one-word paragraph preceded by a paragraph-name.

A paragraph consisting of EXIT may be named as the operand of the THRU option of the PERFORM statement. In this case, a statement in the range of an active PERFORM may transfer control to an EXIT paragraph, thus bypassing the remainder of the statements in the range of the PERFORM. In all other cases, EXIT paragraphs have no effect and control passes sequentially through them.

6.9.3 NOTE Statement

The NOTE statement enables a programmer to write explanatory comments in the Procedure Division of a source program.

The format of the NOTE statement is

NOTE comment .....

e.g.

```
NOTE     that this paragraph creates and outputs to magnetic tape a Master File
record.
```

The NOTE statement is reporduced on the program listing, but has no effect on the content of the object program.

NOTE must begin a sentence. Following the word NOTE, any combination of the characters from the COBOL character set may appear. If NOTE is not the first word in a paragraph, the comments end with a full stop followed by a space. However, if NOTE is the first word of a paragraph, any subsequent sentences within the paragraph are also considered to be comments.

7.0 FAULT FINDING FACILITIES

7.1 TRACE Statement

The TRACE statement indicates the flow of the program at run time.

The format of the TRACE statement is

$$\left[\begin{matrix}\underline{\text{READY}} \\ \underline{\text{RESET}}\end{matrix}\right] \quad \underline{\text{TRACE}}.$$

After a READY TRACE statement has been executed, paragraph and section names are output to the lineprinter each time they are entered. RESET TRACE cancels the effect of the previously executed READY TRACE.

7.2 EXHIBIT Statement

The EXHIBIT statement outputs a literal or the name and contents of a data item to the printer.

The format of the EXHIBIT statement is

$$\underline{\text{EXHIBIT}} \left[\begin{matrix}\underline{\text{NAMED}} \\ \underline{\text{CHANGED}}\end{matrix}\ \left[\underline{\text{NAMED}}\right]\right] \left[\begin{matrix}\text{data-name} \\ \text{non-numeric literal}\end{matrix}\right] \ldots\ldots\ldots$$

7.2.1 NAMED Option

Data-names are displayed on the lineprinter in the form

data-name = value of named item

Literals are displayed on the lineprinter in the form

literal

7.2.2 CHANGED Option

When the EXHIBIT statement is first encountered, at run time, all of the specified data-names are displayed. Subsequently, only those data-names with changed values are displayed. Literals, however, are always displayed.

The format of the display is

value of data-name

or if the value has not changed since the last CHANGED option

number of spaces corresponding to size of data-name

literal

7.2.3 CHANGED NAMED Option

When the EXHIBIT statement is first encountered, at run time, all of the specified data-names are displayed. Subsequently, only those data-names with changed values are displayed. Literals, however, are always displayed. The format for data-names and literals is as for the NAMED option.

7.3 ON STATEMENT

The ON statement permits fault-finding statements to be executed at pre-determined times during a program run.

The format of the ON statement is

ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]

{imperative statement .....} {ELSE / OTHERWISE} {statement ..... / NEXT SENTENCE}

The ON statement is a conditional statement (see 6.3).

When the ON statement is the last clause of a sentence, ELSE (OR OTHERWISE) NEXT SENTENCE may be omitted.

Integer-1, integer-2 and integer-3 must be positive.

Each ON statement has associated with it a compiler-generated counter, which is set to zero when the object program is entered. 1 is added to the counter before the ON statement is executed. When the counter is less than integer-3 and is equal to integer-1, or integer-1 plus any multiple of integer-2, the imperative statement(s) (or NEXT SENTENCE) before the ELSE is executed. If this condition is not fulfilled, the statement(s) (or NEXT SENTENCE) following the ELSE is executed.

(1) If integer-2 and integer-3 are not specified, the condition is fulfilled only when the counter equals integer-1.

(2) If integer-3 is not specified it is assumed to be infinite.

(3) If only integer-1 and integer-3 are specified, integer-2 is assumed to be 1.

APPENDIX 1

RESERVED WORD LIST

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER A
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BEGINNING
BLANK
BLOCK
BY

CALL
CBL-CTR
CF
CH
CHANGED
CHARACTER
CHARACTERS
CHECKING
CHECKPOINT
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORRESPONDING
CPUNCH
CREADER
CREATING
CYCLES

DATA
DATE-COMPILED
DATE-WRITTEN
DATUM
DAYS
DE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DIGIT
DIGITS
DIRECT
DIRECT-ACCESS
DISC
DISPLAY
DISPLAY-ST
DIVIDE
DIVISION
DOWN
DEBUG

ELSE
END
ENDING
ENTER
ENTRY
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXHIBIT
EXIT

F
FD
FILE
FILES
FILE-CONTROL
FILE-LIMIT
FILLER
FINAL
FIRST
FOOTING
FOR
FORM-OVERFLOW
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD

ID
IDENTIFICATION
IF
IN
INCLUDE
INDEX
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS

JUSTIFIED

KEY

LABEL
LABELS
LAST
LEADING
LEFT
LESS
LIBRARY
LIMIT
LIMITS
LINE
LINE-COUNTER

LINES
LINKAGE
LOCATION
LOCK
LOW-VALUE
LOW-VALUES

MINUS
MODE
MORE-LABELS
MOVE
MTAPE
MULTIPLE
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OH
OMITTED
ON
OPEN
OR
ORGANISATION
OTHERWISE
OUTPUT
OV
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PICTURE
PLUS
POSITIVE
PPUNCH
PRINTER
PRINTER 120
PRINTER 160
PREADER
PRINT-SWITCH
PROCEDURE
PROCEED
PROCESS
PROCESSING
PROGRAM-ID
PROTECTION
PUNCH

QUOTE
QUOTES

RANDOM
RD
READ
READER
READY
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELATIVE
RELEASE
REMARKS
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTRICTED
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SD
SEARCH
SECTION
SECURITY
SELECT
SENTENCE
SEQUENTIAL
SET
SIGNED
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STOP
SUBTRACT
SUM
SYMBOLIC
SYNCHRONIZED
SYNSIN
SYSIPT
SYSLST
SYSOPT
SYSOUT
SYSPUNCH

TALLY
TALLYING
TAPE
TAPES
TERMINATE
THAN
THEN
THROUGH
THRU
TIMES
TO
TRACE
TRACK-AREA
TRACKS
TRANSFORM
TRY
TYPE

UNIT
UNIT-RECORD
UNITS
UNTIL
UP
UPON
USAGES
USE
USING
UTILITY

V
VALUE
VALUES
VARYING
VIA

WHEN
WITH
WITHOUT
WORKING-STORAGE
WRITE
WRITE-ONLY

ZERO
ZEROES
ZEROS

Some of these words are included solely for compatibility with 4/50 COBOL.

A P P E N D I X I I

CONTROL LINE

INTRODUCTION

For the COBOL Compiler, the first source line input must be a control line. This line comprises instructions to the Compiler which specify the programmer's requirements for a particular compilation.

A questionnaire, known as a Program Specification Sheet, has been produced to simplify the writing of the control line. When completed, this sheet may be used as a punching document.

Space should be entered for any option or facility that is not required. The number above a box refers to the column number for punch cards.

Description

1 Sequence number

Col. 1 - 6 Value 000000	This entry corresponds to the sequence number of a COBOL source line.

2 COBOL identifier

Col. 8-12 Value COBOL	This entry indicates to the Compiler that this is a COBOL control line.

3 Program name and version number

Col. 13-20	Enter a six character program name and two character version number. Each time the program is compiled, the version number should be updated for documentation and identification. The six characters of the program name are checked against the name in the PROGRAM-ID paragraph in the source program.

4 Source Medium

Col. 21	Enter S if the source program is to be read from magnetic tape (a source master). Enter W if the source program is to be read from a magnetic tape that may be overwritten during the compilation (a work tape).

Enter P if the source program is to be read from punched tape or cards.

5 Source library required

Col. 22

Enter Y if library text is to be included from a magnetic tape source library during compilation.

6 Listing suppression

Col. 23

Enter Y if the listing of the source program and the data map are not required.

Enter D if only the source program listing is required.

Note that the use of either option speeds compilation.

7 Stack object

Col. 24

If the object program is to be written at the end of an existing program file, enter Y, otherwise a new file will be created.

8 Trace

Col. 25

If TRACE statements are to be compiled in the program, enter Y.

9 Load and go

Col. 26

If the program is to be loaded and run immediately after compilation enter Y.

10 Program base address

Cols. 27-32

If the required program base address is not 256 decimal, enter the program base address. The address, which must not be less than 256, must be a decimal integer, right-justified and zero-filled. If no address is entered, the Compiler allocates the the standard base address of 256.

11 Output file name

Cols. 33-44

Enter the name of the program file to which the output is to be stacked, or enter the name to be given by the Compiler to the new program file (see 7).

12 Next do program name

Cols. 45-52

If required, enter the name, and optionally the version number, of the program to be loaded and run after the program to be compiled has been run.

13 Next to $SCC

Cols. 53-56 — Enter the $SCC of the device from which the Next do program is to be loaded.

14 Next do rewind

Col. 57 — If the Next do program is to be loaded from magnetic tape, enter Y to specify rewind before loading.

Note

If a new program file is to be created, no filename need be entered in 11 above. In this case, the Compiler uses the name of the compiled program as the filename.

A P P E N D I X I I I

INDEX	CHAPTER	PAGE
ACCEPT	6.3.6	54
ADD	6.6.2	66
ALTER	6.8.2	79
Arithmetic Expressions	6.2	48
Arithmetic Statements	6.6	64
ASSIGN	4.2	11
AUTHOR	3.0	10
BLOCK CONTAINS	5.1.2	14
Branching Statements	6.8	79
Character Set	2.1	3
CLOSE	6.3.4	53
Compiler Directing Statements	6.9	86
COMPUTE	6.6.1	64
Conditional Statements	6.7	72
Condition Names	5.2.5	22
CONFIGURATION SECTION	4.1	11
DATA DIVISION	5.0	13
Data Items	5.2.6	24
Data Names	5.2.2	18
DATA RECORDS	5.1.6	14
DATE-WRITTEN	3.0	10
DISPLAY	6.3.5	54
DIVIDE	6.6.5	69
Ending Statements	6.5	63
ENTER	6.9.1	86
ENVIRONMENT DIVISION	4.0	11
EXAMINE	6.4.3	60

INDEX	CHAPTER	PAGE
EXHIBIT	7.2	89
EXIT	6.9.2	87
Figurative Constants	5.2.4	21
FILE-CONTROL	4.2	11
File Description (FD)	5.1	14
FILE SECTION	5.0	13
GIVING	6.6.6	70
GO TO	6.8.1	79
IDENTIFICATION DIVISION	3.0	10
IF	6.7.4	78
INPUT-OUTPUT SECTION	4.2	11
Input and Output Statements	6.3	49
INSTALLATION	3.0	10
JUSTIFIED RIGHT	5.3.5	36
LABEL RECORDS	5.1.5	14
Level Numbers	5.2.1	16
Literals	5.2.3	20
MOVE	6.4.1	55
MULTIPLY	6.6.4	68
NEW PAGE	6.3.3.1	52
NOTE	6.9.3	88
OBJECT-COMPUTER	4.1	11
OCCURS	5.3.7	39
OPEN	6.3.1	49
ON	7.3	91
PERFORM	6.8.3	80
PICTURE	5.3.2	26

	CHAPTER	PAGE
PROCEDURE DIVISION	6.0	45
PROGRAM-ID	3.0	10
Punctuation	2.2	5
Qualification	2.4	7
READ	5.3.2	50
RECORD CONTAINS	5.1.4	14
RECORDING MODE	5.1.1	14
REDEFINES	5.3.6	37
REMARKS	3.6	10
ROUNDED	6.6.6	70
SECURITY	3.0	10
SELECT	4.2	11
SIZE ERROR	6.6.6	71
SOURCE-COMPUTER	4.1	11
STOP	6.5.1	63
Subscription	5.3.7.1	41
SUBTRACT	6.6.3	67
TALLY	6.5.3	60
TRACE	7.1	89
TRANSFORM	6.4.4	62
USAGE	5.3.3	34
VALUE	5.3.4	35
VALUE OF ID	5.1.3	14
WORKING-STORAGE SECTION	5.0	13
WRITE	6.3.3	51